Integrating a PCI IP Core to FPGA- Design and Implementation

G.Prasad, N.Vasantha

Abstract - High volume and high throughput rates are the need for high speed data acquisition applications. Higher efficiency and throughput is achieved by the pci bus technology. By becoming a part of the plug & play domain of the host's operating system, no additional data transfer protocols are needed. In this paper we have used a high density field- programmable gate array (FPGA) logic along with PCI master core for high data rate data acquisition. An FPGA with embedded PCI master core serves as a programmable interface between PCI bus and a local FIFO. The application dependent controller functions as well as FIFO and PCI interfacing are handled by FPGA logic. A Linux driver was developed to interface with the core in the FPGA and achieve high bandwidth in DMA mode. This paper will first provide an overview of IP use, including the advantages and disadvantages of using IP. The use of IP will be considered from the view of satellite ground reception applications.

Index terms – Data Acquisition buses, PCI bus, DMA transactions, FIFO read out.

I. INTRODUCTION

PCM data being received from the satellite will be of large volume and since the data rate of the incoming data is very high an interface to the host which will meet the demands of high throughput and have a roadmap for the future is required.

a. IP CORE PCI_MT64 MEGACORE FUNCTION used in our design.

The IP core provides an interface between the Altera pci_mt64 Mega Core function and a 64-bit, 2-MByte FIFO module [12]. It Supports 32- and 64-bit PCI master and target transactions, Supports chaining and non-chaining mode DMA, Uses the dual-port FIFO buffer function from the library of parameterized modules (LPM), This design shows how to connect the local-side signals of the Altera pci_mt64 Mega Core function to local-side applications when the Mega Core function is used as a master or target on the PCI bus. The design consists of the following elements Master control logic, DMA engine, Data path FIFO buffer functions and FIFO interface as shown in

Manuscript received October, 2013.

G.Prasad, Scientist "SF", National Remote Sensing Centre, ISRO, Hyderabad, India.

N.Vasantha working as Professor & Head, Department of Information Technology, Vasavi College of Engineering, Hyderabad, India.

Fig. 1.

When the pci_mt64 function is acts as a master, the master control logic interacts with the DMA engine to control the PCI master transactions. During a PCI master write, the data flows from the local master to the PCI bus.

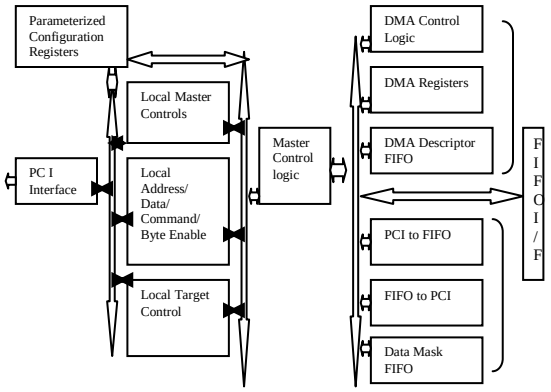

FIG 1. Block Diagram of PCI –Mt64 IP Core

The master control logic Provides status of the PCI bus to the DMA engine, Interacts with the pci_mt64 function to execute a PCI master write cycle, Transfers the data from the external FIFO-to-PCI FIFO buffer to the pci_mt64 function. The DMA engine interfaces with the master control logic, the data path FIFO buffer s, and the FIFO interface to coordinate DMA transfers to and from the FIFO. The DMA engine consists of DMA control logic, DMA registers, DMA descriptor FIFO buffers. The DMA control logic Provides control signals to the master control logic to prompt it to request the PCI bus when needed, Triggers a new access to the external FIFO, Monitors the data path FIFO buffer's and the current FIFO access, Monitors the DMA registers in order to initiate a new transaction, Loads the address counter register (ACR) and byte counter register, (BCR) in the DMA registers when DMA is in chaining mode, Updates the interrupt status register (ISR) and control and status register (CSR) in the DMA registers (chaining and non-chaining mode). Setting up the DMA registers in the DMA engine initiates DMA transactions.

The DMA descriptor FIFO buffer provides the storage area for the series of byte count and PCI address pairs when the DMA is programmed to operate in chaining mode. The size of the descriptor FIFO buffer is 256 x 32, and it is capable of holding up to 128 DMA transactions in a chain. This FIFO buffer must be written with byte count and address pairs by another master/host on the PCI bus before starting the DMA in chaining mode. The descriptor FIFO buffer is read by the DMA control logic to fetch the current byte count to the BCR and address to the ACR before executing the next DMA

transaction in a chain. The data path FIFO buffers serve as the buffer space for the data flowing between the external FIFO and PCI bus. The FIFO buffers are needed to resolve the external FIFO's high data-access latency.

II. FPGA 90NM STRATIX EP1S25F1020C5

The Stratix FPGA [15] is used to implement the four modules i.e. Data Simulator, on chip memory, fifo read/write & control logic and PCI IP Master core. Stratix devices contain a two-dimensional row- and column-based architecture to implement custom logic. A series of column and row interconnects of varying length and speed provides signal interconnects between logic array blocks (LABs), memory block structures, and DSP blocks. The logic array consists of LABs, with 10 logic elements (LEs) in each LAB. An LE is a small unit of logic providing efficient implementation of user logic functions. LABs are grouped into rows and columns across the device. M512 RAM blocks are simple dual-port memory blocks with 512 bits plus parity (576 bits). These blocks provide dedicated simple dual-port or single-port memory up to 18-bits wide at up to 318MHz. M512 blocks are grouped into columns across the device in between certain LABs. M4K RAM blocks are true dual-port memory blocks with 4K bits plus parity (4,608 bits). These blocks provide dedicated true dual-port, simple dual-port, or single-port memory up to 36-bits wide at up to 291MHz. These blocks are grouped into columns across the device in between certain LABs. M-RAM blocks are true dual-port memory blocks with 512K bits plus parity (589,824 bits). These blocks provide dedicated true dual-port, simple dual-port, or single-port memory up to 144-bits wide at up to 269MHz. Several M-RAM blocks are located individually or in pairs within the device's logic array.

III. INTELLECTUAL PROPERTY OVERVIEW

Currently FPGAs, particularly RAM based devices such as Altera Stratix, are large enough that they can hold significant system components, such as bus interfaces, processors, DSP blocks, communication standards and large blocks of custom logic. However, to use the ocean of resources available in the FPGA by developing system components, in house takes a large design team with expertise in a range of areas, as well as an overwhelming amount of testing. For this reason IP is now widely used method of FPGA development, where definitions of system components at some stage in the implementation flow can be purchased for use in FPGA. In fact multiple components can be integrated with custom logic on an FPGA to build an entire system on a chip (SOC)[8]. Intellectual property for FPGAs generally comes in two different forms, as an encrypted net list (generally synthesized prior to encryption) which effectively prevents the design from being modified (it is encrypted to prevent sections of the IP from being copied), or as a register- transfer –logic (RTL) HDL description that can be modified. These types of cores are known as "soft", since they must still undergo the place and route portion of the FPGA implementation flow. At times, high performance cores (such as optimized processors) will comes as a post place and route net list that is designed to fit in a particular section of an FPGA. This is known as a "hard", core since its location with the FPGA is already determined.

A. Advantages of Intellectual Property.

The advantages of using IP are a) reduce design risk, b) shorter test and debug cycles, c) increased integration d) design reuse. These translate to additional advantages for proto hardware. E) lower poser and smaller size due to increased integration. F) smaller form factors due to shrinking size of FPGAs. G)Future reusability, which can eliminiate heritage issues.

B. Disadvantages of Intellectual Property.

The main disadvantage of using IP is that in encourages overly aggressive scheduling. IP is often advertised as offering a nearly turn –key solution to design problems, and in fact it can be in common place situation. In general, IP uses requires a large amount of knowledge of the core's technology, time to become familiar with a particular core's operation and time to test the final FPGA. For mission critical applications an additional disadvantage of using IP is in the effort required to evaluate if a core is safe for that requirement and place. In case of applications like space the IP must be as single event upset immune as possible, with safe state machines, registers that refresh and triple voted flip-flops. Some of these features may be provided by the FPGA being used.

IV. DATA ACQUISITION SYSTEM

In our design of the data acquisition system hard IP core has been used and integrated with the Stratix FPGA and the remaining logic is developed as shown in the block diagram Fig 2.

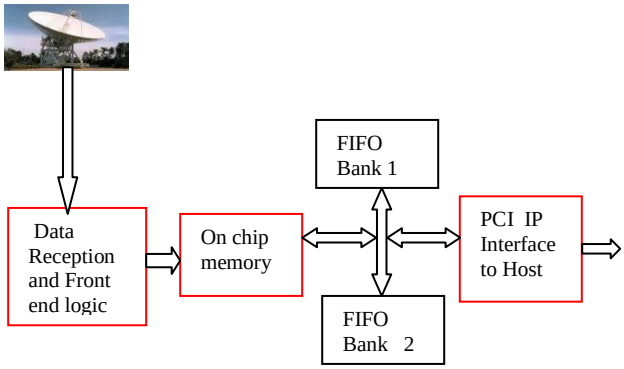

Fig 2.The blocks in the above diagram with red color are implemented in FPGA.

A. Data Simulator

The Data Simulator is to simulate the data. Data Simulator logic generates the FS code, variable line count in the aux field and fixed video data pattern. A Crystal Oscillator of 200MHz is the source which is divided to generate the required frequency. The serial data and clock are connected to a RJ45 connector.

B. On Chip Memory and Fifo Write/ Read.

Simulated data and clock are inputted to the frame synchronizing logic, where the valid frames are detected. After frame synchronization, individual measurands are identified according to the frame location. The decommutator identifies and extracts embedded asynchronous data stream (EADS) words. The serial data is converted to 64bit (Qword)

parallel data. In our design the on chip memory was designed using the M 4K RAM block of the Stratix device. The FPGA memory was designed to store 1KQWords (Qword =64bit) and as two banks. Using the alternate buffer concept as shown in Fig 3 the read and write operations were designed so that 1000 words i.e. 3 frames (each frame is 302 Qwords) of data can be written in one buffer and then the write operation will switch to the next buffer and remaining 1000 Q Words will be written. When the first bank writing is complete a control signal will be issued by the memory controller so that the first buffer data can be read on to the external FIFO, thus the 3 frames written in the first bank will be written to the external FIFO by enabling the write control signal and write clock of the external FIFO. After read out from the first on chip memory bank now this bank will be ready to take in the next 3 frames. Next the second memory bank will be writing to the external FIFO and this memory will also be available for next 3 frames. This will result in continuous write and read between on chip memory and external FIFOs. Thus a large memory of desired volume can be realized. Read operation of the external FIFO will empty the FIFO thus in this way required volume of data from multiple inputs can be written and stored.

The buffer logic consists of two 128K X 72bit per channel in depth expansion mode. The FIFO's used are IDT72T72115L10BB. The 64bit Parallel data is written into FIFO and when the FIFO is Half Full data ready status will be indicated to the PCI logic and read logic will be initiated. Thus the read will be taking place continuously since the data is present in the FIFO always.

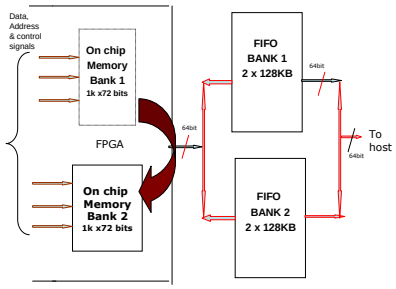

Fig 3. Memory implementation on FPGA and interface to external FIFO.

Result: Total Memory bits that were required to realize the above design was 573456bits out of 1944576 bits (i.e. 29.49% of the memory bits) and since toggling between high frequency clock (read) and low frequency clock (write) a power saving of 10% was observed.

C. PCI Interface logic.

The IP core used in our design implements two base address registers (BARs)[12]: BAR0 and BAR1. BAR0 is set to reserve1 MByte of system address space to map all the local read/write registers and the descriptor FIFO buffer. The 12 most significant bits of the PCI address are decoded by the IP to claim a target transaction. All of the DMA registers and the descriptor FIFO buffer are mapped to this address space. This reserved space is divided into two equal regions of 512 KBytes: the upper memory region is used for the DMA FIFO buffer and the lower region is used to map the DMA registers. BAR1 is set to reserve 16 MBytes of the system address space to map to the lower 16 MBytes of the external FIFO.The control and status register CSR is a 9-bit register and is used to configure the DMA engine. It directs the DMA operation and provides the status of the current memory transfer. The CSR can be written/read by another master on the PCI bus. The address counter is a 32-bit register. The ACR is implemented with a 30-bit counter and the 2 least significant bits are tied to ground. The ACR contains the PCI bus address for the current memory transfer and is incremented after every data phase on the PCI bus. The PCI bus memory transfer initiated by the DMA engine must begin at the DWORD boundary. The ACR can be written/read by the PCI bus master. The ad_loaded bit triggers the beginning of the DMA operation because it sets the dma_on bit in the CSR. It is automatically set when the write to the ACR occurs. Therefore, the ACR must be written last when setting up the DMA register. The DMA byte counter is a 18-bit register. The BCR is implemented with a 15-bit counter and the 2 least significant bits are tied to ground. The BCR holds the byte count for the current DMA memory transfer and decrements (by 4 bytes) after every data transfer on the PCI bus. The BCR can be written/read by the PCI bus master. The interrupt and status register is a 6-bit register. The ISR provides all interrupt source status signals to the interrupt handler. The ISR is a read only register and can be read by another master on the PCI bus

D.DMA Engine

Idle: The channel is idle when no DMA is under progress. A transfer can be programmed and writing to DMA command fields starts a new transfer.

Busy*:* DMA Engine is busy while processing data transfer. It remains so until transfer is either complete or stopped.

Complete*:* When all the data has been transferred, transfer is complete and channel becomes idle on the next clock cycle.

Stop/Error: Transfer ends immediately if a master abort termination occurs during a transfer or if an error is detected in the page descriptor chain.

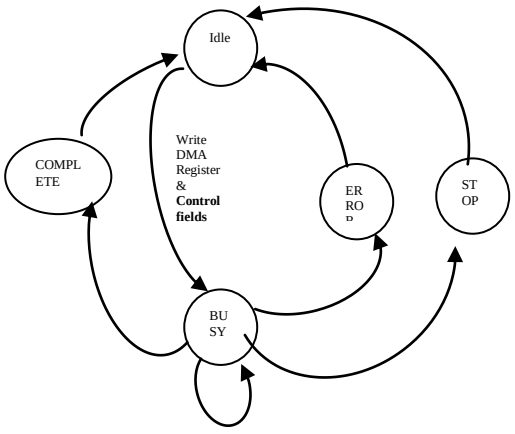

FIG 4. DMA STATE DIAGRAM

E. Non-Chaining Mode

To operate the DMA in non-chaining mode, the DMA registers must be written in the following sequence:

Write the CSR register with the appropriate value (chain_enablebit = 0).Write the BCR with the number of bytes to be transferred. Write the ACR with the starting PCI address for the transaction.

After the ACR is loaded with the PCI address, the ad_loaded bit in the ISR register is set and the DMA state machine is triggered. In the subsequent clock cycle, the dma_on bit of the CSR register is set to indicate that the DMA transfer is in progress. The DMA state machine then sends the request to signal the master control logic to request a PCI master transaction, the master control logic forwards the request to the IP core function interface to request the PCI bus. The DMA state machine also asserts the local start signal to request an FIFO access through the FIFO control logic.

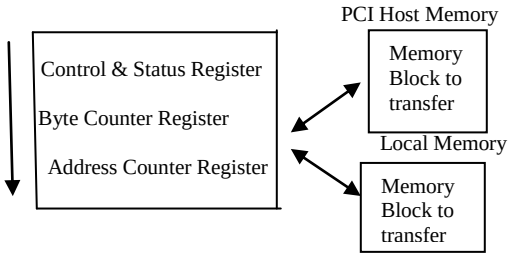

Fig 5. Non chaining Setup sequence

Once the bus has been granted to the IP master, data transfers can take place if data is available in the appropriate data FIFO buffer. If data is not available, wait state(s) will be inserted on the PCI bus. The BCR counts down after every data transfer on the PCI bus for DMA write and on the local side for DMA read until it reaches zero. The DMA control logic then sets the dma_tc and int_pend bits and resets the ad loaded and dma_on bits to return the DMA to the idle state. In the event of an abnormal termination of a DMA transaction where the byte counter has not expired, the DMA state machine waits for the master and FIFO control logic to return to the idle state before requesting a new DMA transaction to transfer the remaining data.

F. Chaining Mode

To operate the DMA in chaining mode, the descriptor FIFO buffer and the DMA registers must be written in the following sequence:

Target burst write the series of DMA transfer information to the DMA descriptor FIFO buffer. The DMA transfer information contains the PCI address and byte count to be transferred.Write the appropriate control data bits (chain enable bit = 1) to the CSR.When the chain enable bit is set, the start chain bit of the ISR will be set to indicate that the DMA is in the chain operation mode. This action triggers the loading of the first set of DMA data from the descriptor FIFO buffer to the ACR and BCR. After the ACR and BCR are loaded with valid data, the DMA state machine requests a DMA transaction. The DMA then operates similar to the non-chaining mode.

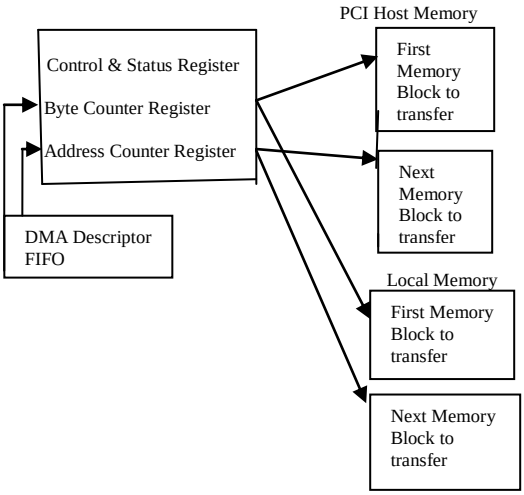

Fig 6. Chaining DMA Setup sequence

After the first DMA transfer has completed and the next set of DMA data is loaded from the descriptor FIFO buffer into the ACR and BCR, a new DMA transfer takes place. This process is repeated until the DMA FIFO buffer becomes empty, i.e., the DMA chain has ended. The DMA control logic then sets the dma_tc and int_pend bits and resets the ad_loaded and dma_on bits to return the DMA to the idle state.

In our design the chaining and non chaining modes were tested. Since the Byte counter register is 15bits i.e. it can accommodate up to 32Kbytes of data in one non chaining mode. Since the data being acquired from the satellite is in Giga Bytes chaining mode was tested. The DMA descriptor FIFO is loaded with the value so that required Giga Bytes of Data is acquired in DMA mode. The serial data from two channels from the Data simulator/satellite after frame sync detection is converted into 64bit (Qword) parallel data. Parallel data from the FIFO's are read out to the Host Server through this PCI Interface logic operating at 66MHz. The 66MHz clock is derived from the Host server. The 64 bit data read out from the two channels will be written into two files. The Embedded Hardware is integrated with Higher end server with Linux as Operating System.

Since the BCR register can hold 32Kbytes and there are two channels to be read through the PCI core, the 32Kbytes are divided into two i.e. each channel will get 16Kbytes . Thus data reads through the PCI core will be toggling between 16Kbyte boundaries in chaining mode till the total required amount of data is transferred to the host.

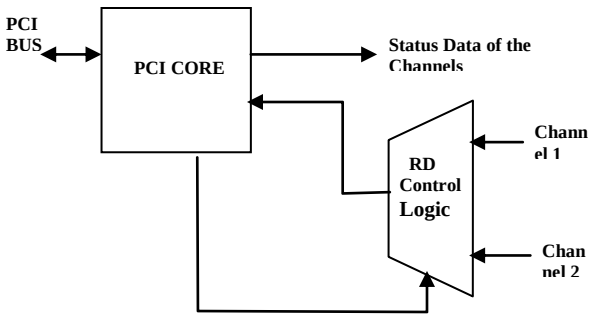

Fig 7 . PCI Interfacing diagram.

The read control logic will start one channel read out and place it on the PCI bus as input to the core as shown in Fig 7.. After transferring the required number of bytes as mentioned in the Byte Counter Register an acknowledgement signal from the DMA engine to the read control logic will switch to the second channel and data will be sent as input to the PCI core, this process will continue till the required amount of data is read out. The PCI core interface is integrated to the developed driver in VC ++ and the testing was done on an Itanium server. The throughput achieved is 220Mbytes per second.

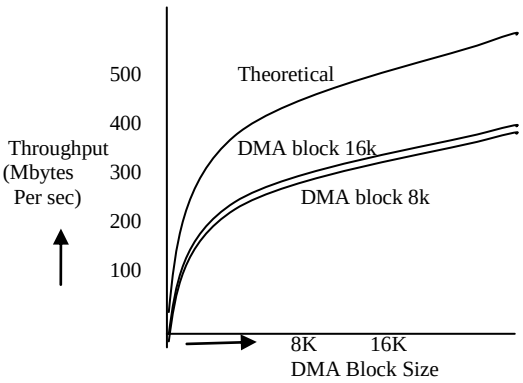

Fig.8 Throughput for different DMA blocks

Result: 5777 logic elements i.e. 23% of the Stratix FPGA capacity and 318 I/Os i.e. 45% of the FPGA I/O were used to implement the pci interface. Through put of 220Mbytes/ second was achieved. Since the quartus structure file of the core was used directly the routing of all the signals was optimal and hence a power saving of 15% was achieved

V. SIMULATION RESULT

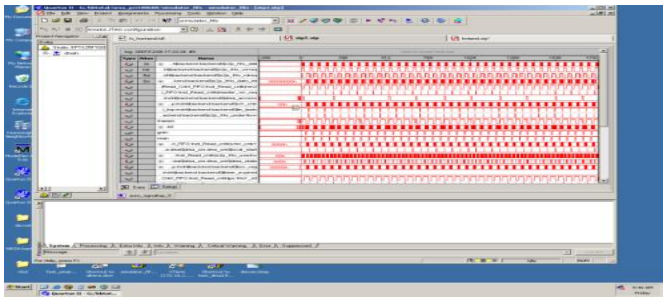

Fig 9 . DMA burst in chaining mode

VI. CONCLUSION

The FPGA based PCI core interface, and associated logic designed and developed is suitable for satellite data acquisition systems in the Ground segment. Since the hardware is compact and can be housed in any true 64bit server this forms a embedded hardware and is thus suitable for fixed and mobile applications also. The throughput to the host achieved is between 100 to 220Mbytes/second thus it can cater to high speed data acquisition. Since the major modules are incorporated into the FPGA a power reduction of nearly 20% is achieved in the design. The logic is validated with an inbuilt simulator so that the total chain involved in the design is completely tested.

REFERENCES

[1] J. Toledo, H.Muller, J. Buytaert, F.Bal, A.David, A.Guirao and F.J.Mora (2002), "A plug and play approach to data acquisition", Network architecture and performance digital equipment corporation, Littleton
[2] Charles Geber, Kevin Yee (2000), "Peripheral component interfaces with quick logic QL1624 FPGA", quick logic corporation, Santa Clara.
[3]. Jim McManus, PCI Applications Engineer, Xilinx Inc " Using FPGAs as a flexible PCI Interface Solution".
[4]. P.Assis, P.Brogueira, L.Melo, M.Pimenta, J.c.Silva, J.Varela LIP-Lisbon, Portugal(2003), 28th International Cosmic Ray Conference. " A PCI based data acquisition system for Ground Array Detectors with Wireless Synchronization Through GPS".
[5]. David Robinson, Patrick Lysaght, Gordon M cGregor and Hugh Dick " Performance Evaluation of a Full Speed PCI Initiator and Target Subsystem using FPGAs".
[6]. Daniel Zisner, Jurgen Teich " Power Signature Watermarking of IP Cores for FPGAs.
[7]. Toledo, J. Dept. of Electron. Eng., Univ. Politecnica de Valencia Muller, H Buytaert, J. Bal, F David, A Guirao, A.MoraF.June 2002. " A Plug and Play to Data Acquisition".
[8]. Haber .J (2003)." Using a commercial IP core in space flight avionics. Lessons learned".
[9]. Pillem Ramesh, Venkata Aravind Bezawada, K S N Vittal, Dr. Fazal Noorbasha (2012)." Design of 64-bit Peripheral Component Interconnect Bus at 66MHz".
[10]. Nupur Shah, Design Engineer, Xilinx "The Challenges of Doing a PCI Design in FPGAs".
[11] S. Palanivelu, J.Shanmugam Prof and Head, Division of Avionics, Madras Institute of Technology, Chrompet, Chennai "Design and Development of PCM Decommutator with PCI –Interface."
[12] Altera Master Core IP Mt64 User Guide.
[13] On-Chip FIFO Memory Core in Volume 5: Embedded Peripherals of the Quartus II
[14] Memory System Design from Altera Corporation February 2010.
[15]. Stratix Device Handbook, Volume 1 July 2005.

G.Prasad (M-IEEE, FIETE) received M.Tech degree in Electronics from JNTU Hyderabad in 1995, MBA from IGNOU, New Delhi 2000. He is presently working as Scientist "SF" at National Remote Sensing Centre, ISRO, Hyderabad. His nature of work includes design and development of satellite data acquisition systems, high speed communication between different data acquisition sites through satellite networking. His research interests include VLSI designs, embedded system and realizing systems on programmable chips.

N.Vasantha (M-IEEE, LM-CSI,IETE,ISTE, M VSI) received the B.E. degree from College of Engineering, Guindy, Madras, in 1977, the M.Tech. degree from JNTU, Hyderabad, AndhraPradesh, in 1986, the Ph.D. degree in Electronics & Communication Engineering from the Osmania University, Hyderabad, AP, in 2008.She is currently working as Professor & Head, Department of Information Technology, Vasavi College of Engineering, Hyderabad, AP. She is the recipient of the prestigious IETE-Prof. K.Sreenivasan's Memorial Award(2010)