ISSN 2250-3153

FPGA based hardware as a coprocessor

G.Prasad, N.Vasantha

Abstract- The process of hardware – software co-design of satellite data acquisition system is described. The hardware components are targeted to execute on a reconfigurable hardware coprocessor which communicates with a host computer that executes the software tasks. Control of the data flow between device interfaces, processing blocks and memories in a data acquisition system is complex in hardware implementation.. With growing computational needs, high design and NRE costs of ASICs, FPGA based co-processor has become a viable alternative. The main objective of our co-design methodology is the usage of hardware designing of algorithms, simulation and synthesis.

Index Terms- PCI Core, FIFO, DMA, Derandomization etc.

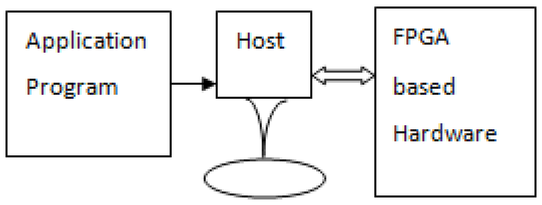
I. Introduction

Over the past decade, high performance computing (HPC) applications demands have out spaced the conventional processors performance by demanding more processing power. As a result hardware acceleration became necessary to augments processors with application specific coprocessors. Typically, co-processors, such as graphics co-processors, are application-specific. The computations they perform are predetermined. Co-processors were initially designed to reduce the computational overload on the host processors, thus scaling-up the latter's performance. Due to right combination of performance, ease of use price along with significant power savings. FPGA is a good option for coprocessor design

With the advent of Field –programmable device technologies, reconfigurable coprocessors have become a viable alternative to quickly and inexpensively realizing quick implementations of wide classes of applications. In this paper the reconfigurable coprocessor board comprising of one Field Programmable Gate arrays (FPGA), First in First Out (FIFO) and PCI-X logic to interface with the host system was designed and is discussed. Using memory mapped addressing. The host computer can access the FIFO on the board through the PCI-X interface and send control words to the FPGA hardware. The functions to be implemented on the coprocessor board are selected from the software that is already written for and executing on the host computer.

Selection of these functions, which we will call hardware functions, is motivated by the desire to achieve overall speedup and is constrained primarily by the size of the FPGA. Hardware design and implementation is intensive in design time. Therefore proper choice of hardware functions is critical. Rapid prototyping and performance evaluation of the hardware functions is crucial to the successful and efficient use of coprocessor technologies.

In this paper we will describe an approach to the selection and rapid prototyping of coprocessor function. The design process starts with the existing software implementation of the algorithm. This software implementation is then profiled for time and critical portions of software where most of the CPU time is spend are identified. This data is used to select the functions for migration to hardware. After having isolated the functions to be implemented on hardware, next an algorithmic behavioral specification for the hardware task is developed in HDL and is thoroughly verified and simulated for proper functionality. We will describe our co design through an example implemented in the data acquisition system.



User program Host Processor FPGA Coprocessor

Fig 1. FPGA Coprocessor

II. Quartus Software and Altera Hardware Descriptive Language

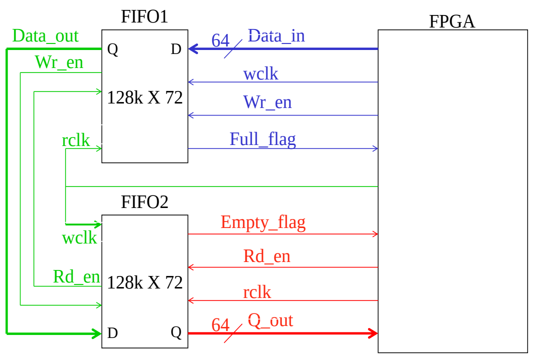
The Altera Quartus II design software is a multiplatform design environment that easily adapts to specific needs in all phases of FPGA and CPLD design. Quartus II software delivers the highest productivity and performance for Altera FPGAs, CPLDs, and Hard Copy ASICs. Quartus II software delivers superior synthesis and placement and routing, resulting in compilation time advantages. Compilation time reduction features include, Multiprocessor support, Rapid Recompile, Incremental compilation. Quartus II Analysis and Synthesis, together with the Quartus II Fitter, incrementally compiles only the parts of your design that change between compilations. By compiling only changed partitions, incremental compilation reduces compilation time by up to 70 percent. For small engineering change orders (ECOs), the Rapid Recompile feature maximizes your productivity by reducing your compilation time by 65 percent on average, and improves design timing preservation.

AHDL is a proprietary digital Hardware Description Language (HDL) from Altera Corporation for programming their Complex Programmable Logic Devices (CPLD) and Field Programmable Gate Arrays (FPGA). This language has an Ada programming language-like syntax and similar operation to VHDL or Verilog. It is supported by Altera's Quartus and Max+ series of compilers. An advantage of AHDL is that all language constructs are synthesizable. AHDL is to Verilog much as assembly language is to a higher-level programming language: in AHDL, you have more control.

III. ARCHITECTURE

This section gives an overview of the proposed architecture of the FPGA based hardware which will perform the function of the coprocessor
The architecture consists of three major blocks
1. FIFO Bank
2. A PCI –X core interface
3. 90nm FPGA.

A block diagram of the proposed architecture is shown in **Fig 2**. The entire system sits on a PCI card which can be fitted to a server. The system can be operated by issuing instructions to the FPGA through the PCI interface.



FIFOs in depth expansion fashion

Fig 2. FIFO Bank

The term external FIFO refers to any device that you connect externally to a FPGA. There are several varieties of external FIFO devices. The choice of external FIFO and its type depends on the nature of the application. Designing with FIFO memories presents both advantages and disadvantages. External FIFO devices provide larger storage capacities than on-chip memories, and are still quite fast, although not as fast as on-chip memories. Typical external FIFO devices have capacities ranging from around 128 Kbytes to 10 Mbytes.

External FIFOs are typically very low latency and high throughput devices, slower than on-chip memory only because they connect to the FPGA over a shared, bidirectional bus. FIFO devices are less expensive per Mbyte than other high-capacity memory types such as SRAM. The primary advantages of external FIFO in an FPGA-based embedded system are cost and board real estate. They also consume less board space per Mbyte than SRAM and FPGA on-chip memory, which consumes none.

Depending on the available on chip memory the FPGA memory can be partitioned to fit in data corresponding to a fixed number of words. The on chip memory width is set up to match the width of the external FIFO which is the off chip memory. This number of words can be interfaced to the external onboard FIFO and depending on the size of the FIFO large volume of data can be stored. Thus by doing this framing accuracy as generated in the RAM is maintained. This is illustrated below as how it was implemented in an FPGA in the data acquisition hardware developed. 128k X 72 bit FIFOs are used in the depth expansion fashion, to create one 256k X 72 bit FIFO. Therefore, total 4 FIFOs are required to create 256k X 72 bit FIFO per data channel.

IV. IP CORE PCI_MT64 MEGACORE FUNCTION

The IP core provides an interface between the Altera pci_mt64 MegaCore function and a 64-bit, 2-MByte FIFO module. 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 MegaCore function to local-side applications when the MegaCore 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 Fig 1.

A. Master Control Logic

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. 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.

B. DMA Engine

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.

C. DMA Control Logic

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).

D. DMA Registers

Setting up the DMA registers in the DMA engine initiates DMA transactions. These registers are memory-mapped to BAR0 of the pci_mt64 function; they can be accessed with a target transaction to their memory-mapped addresses. The registers must be written by another master on the PCI bus. The DMA registers consists of Control and status register (CSR), Address counter register (ACR), Byte counter register (BCR), Interrupt status register (ISR), Local address counter.

E. DMA Descriptor FIFO Buffer

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.

F. Data Path FIFO Buffers

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. The design implements the following FIFO buffer's PCI-to-internal FIFO buffer (128 x 64), Data mask FIFO buffer (128 x 8).

The PCI DMA core is interfaced with the host through the PCI bus, The PCI interface is responsible for accessing the configuration parameters to and from the system. The PCI core takes inputs for the functions from the host and stores them in the local configuration registers. Also Data from the external system will pass through the FIFO banks, through the PCI core to system memory. Data from the hard disk is read out by the core, stored in the onboard FIFOs and read out for interface to external devices.

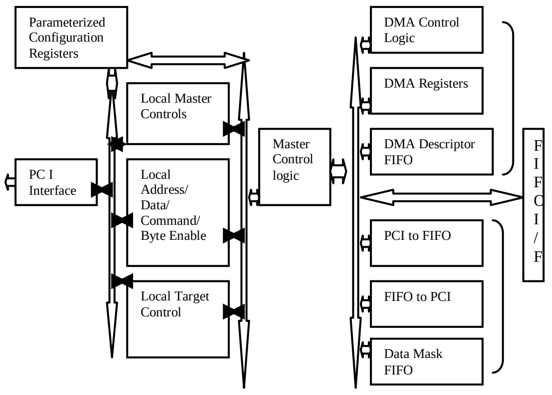


Fig 3. Block Diagram of PCI –Mt64 IP Core

V. FPGA 90nm Stratix EP1S25F1020C5

The Stratix FPGA is used to implement the following modules. Static configuration of on chip memory and PCI core, interface & control logic, and Dynamic Configuration logic. Stratix devices contain a two-dimensional row- and column-based architecture to implement custom logic [19]. 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 318 MHz. 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 291 MHz. 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 269 MHz. Several M-RAM blocks are located individually or in pairs within the device's logic array.

Digital signal processing (DSP) blocks can implement up to either eight full-precision 9 × 9-bit multipliers, four full-precision 18 × 18-bit multipliers, or one full-precision 36 × 36-bit multiplier with add or subtract features. These blocks also contain 18-bit input shift registers for digital signal processing applications, including FIR and infinite impulse response (IIR) filters. DSP blocks are grouped into two columns in each device. Each Stratix device I/O pin is fed by an I/O element (IOE) located at the end of LAB rows and columns around the periphery of the device. I/O pins support numerous single-ended and differential I/O standards. Each IOE contains a bidirectional I/O buffer and six registers for registering input, output, and output-enable signals. When used with dedicated clocks, these registers provide exceptional performance and interface support with external memory devices such as DDR SDRAM, FCRAM, ZBT, and QDR SRAM devices. High-speed serial interface channels support transfers at up to 840 Mbps using LVDS, LVPECL, 3.3-V PCML, or HyperTransport technology I/O standards.

VI. DESIGN AND IMPLEMENTATION

The FPGA hardware board sits on a PCI-X interface bus which can be fitted into a standard server class machine. The data acquisition hardware can be accessed by issuing instructions through the drivers to the PCI master core for selecting the required configuration. The FPGA based hardware interfaces with the on board FIFO bank for reading data from it and transfers the data to host through the PCI –X interface. Also communication from the host for configuring the hardware is also done through the PCI-X interface for selecting the required inputs as defined by the application program.

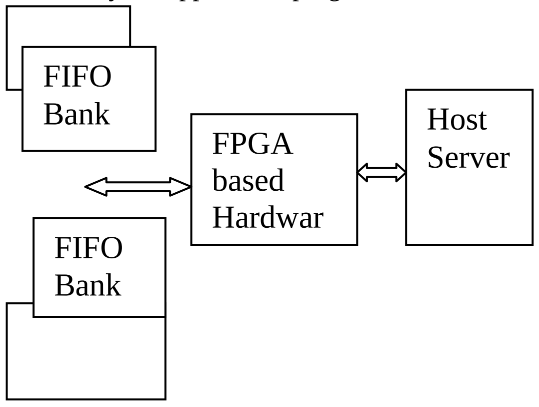


Fig 4. Elements of FPGA Data Acquisition system Architecture.

The hardware architecture is shown above. Various parameters are selectable like, type of sensor, resetting the device, volume of data to be acquired, chaining mode or non chaining mode, DMA Page size etc. needs to be given as input to the FPGA. By selecting the parameters through the application software the FPGA is loaded and only a portion of the logic space on the FPGA is reconfigured. The configuration of the remaining portion of the FPGA is untouched and hence, the functions present in the untouched logic space may be accessed for execution is selected for execution.

DAQ application is a program used to control the FPGA based hardware. It is a program used to control the FPGA device and capture data from the hardware. The Steps to use Nrsa_daq_application are as follows:

Execute the application by issuing typing command Nrsa_daq_application in the command prompt

A list of menu options are seen, these options will

- Reset the device.
- Load parameters to the DAQ hardware
- Select the option "Set DMA Page Size",
- Select the option "Set DMA Transfer Size
- Chaining/Non Chaining mode
- Derandomization on/ Derandomizatiion off.
- Memory read/Memory write

The data acquisition hardware is first reset so that the registers corresponding to the DMA transfer are initialized to the start of value. The parameters for a particular satellite i.e. the frequency of operation, number of sensors, number of data acquisition cards on the server is given as input for the hardware to be configured to acquire the selected satellite. The page size is a customized field and can be modified as per our requirement. Default Page Size is 16KB.The transfer can be done only with a minimum of 1 MB. User can also set a required value. Whether the hardware is configured to handle small chunk or large chunk of data is decided by the Chaining/Non chaining mode selection.
If the derandomization enable is set to ON then data is derandomized in hardware and is acquired. If this parameter is OFF than the derandomization is done in software. Selecting the Memory Read option the transfer takes place and the resulting data will be present in hex. out file.

By selecting the parameters the FPGA is loaded with the corresponding inputs and it is configured for data acquisition through the PCI interface. Now the FPGA processes these parameters as per the configuration logic and also does the processing like Derandomization there by achieving the real time speed. Thus the FPGA off loads the main processor by doing the derandomization process in hardware thus achieving real time speed which is not the case when the derandomization was done in software. Flow diagram of Derandomization in the software is shown Fig 5.

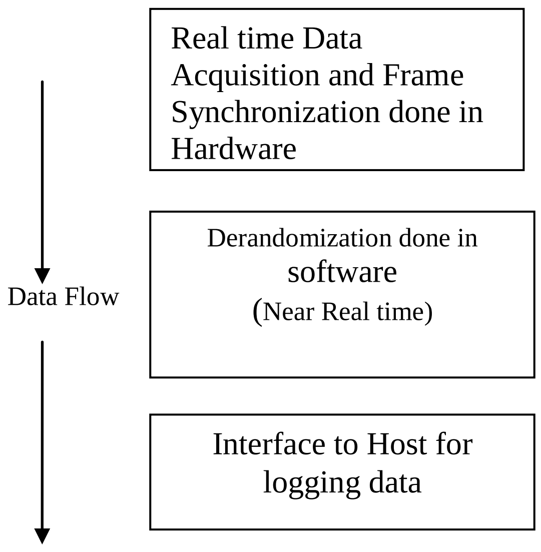

Real time Data Acquisition and Frame Synchronization done in Hardware
Derandomization done in software (Near Real time)
Interface to Host for logging data
Data Flow

Fig5. Data Flow diagram of Software Derandomization

In software the satellite data is frame synchronized and acquired in real time. The further processing of Derandomization is carried out using an application program in near real time. Then the derandomized data is transferred to FIFO onboard memory for further transfer to host system. Thus there is latency in deriving the derandomized data by software method. In this method the host processor is involved in derandomization process of the raw data acquired.

Implementation in FPGA does the frame synchronization, derandomization in real time as shown in Fig6. The logic is realized using a 11bit shift register where the clock is connected to the global clock and reset to the global reset pin. The 11bit shift register is loaded with the control word on power on and the 12th bit is achieved by the exoring of the 8th and 10th bit. The output of the 10th bit is the derandomized serial pattern which is again added to the incoming satellite data to obtain the derandomized output data.

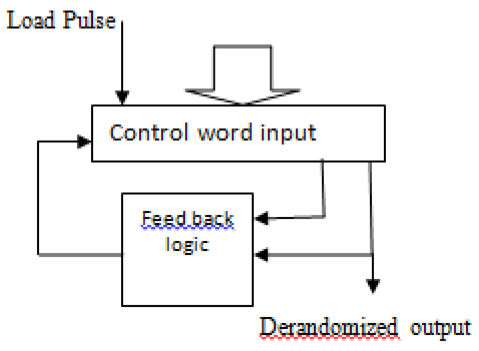


Fig 6. Block diagram of Derandomization principle.

HDL code for implementing Derandomization is given below.

```
data_sim_pn_shft_reg[].clk=GLOBAL(channel_clk);
```

data_sim_pn_shft_reg[].clrn = GLOBAL(nReset);

data_sim_pn_shft_reg[].d=B"00001111011";

data_sim_pn_shft_reg[0].d=(data_sim_pn_shft_reg[8].q $

data_sim_pn_shft_reg[10].q);

data_sim_pn_shft_reg[10..1]=data_sim_pn_shft_reg[9..0]; -- pn sequence shift register enable.

pn_seq_op[0] = data_sim_pn_shft_reg[10];

pn_seq_op[1] = pn_seq_op[0];

Depending on the configuration selected the control word will change and also the load pulse of the control word will vary. This part of the logic space on the FPGA is reconfigured. The remaining portion of the FPGA is untouched and hence the functions present in the untouched logic space may be accessed for execution. The flow chart in Fig 7. shows the sequence of events that take place in the FPGA for implementing the derandomization process in real time in the FPGA.

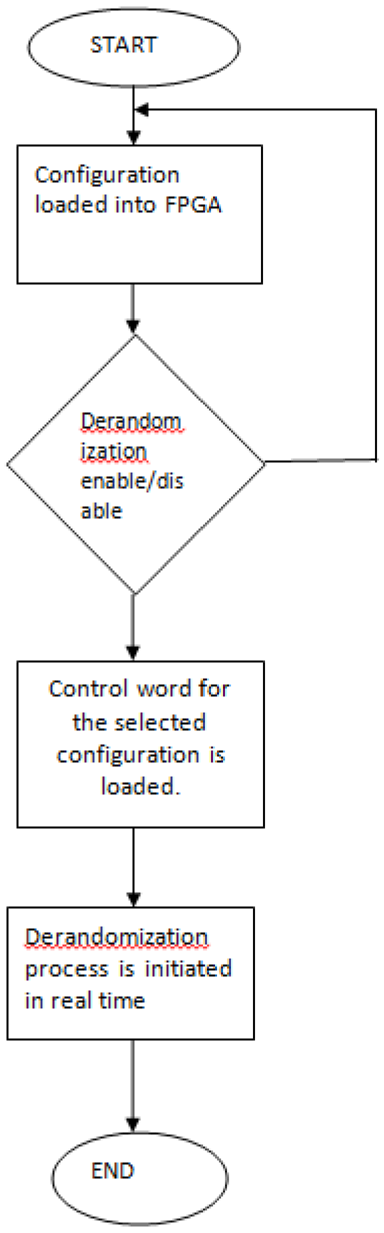

START
Configuration loaded into FPGA
Derandom ization enable/dis able
Control word for the selected configuration is loaded.
Derandomization process is initiated in real time
END

Fig 7. Flow chart of Hardware Derandomization

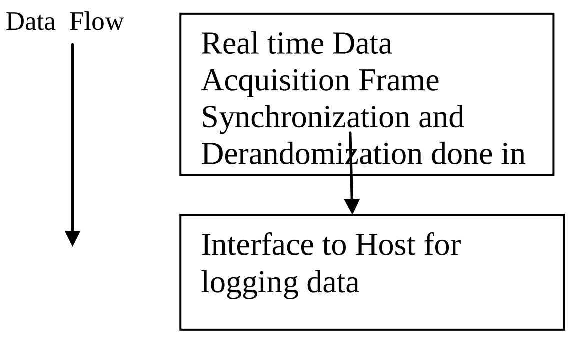
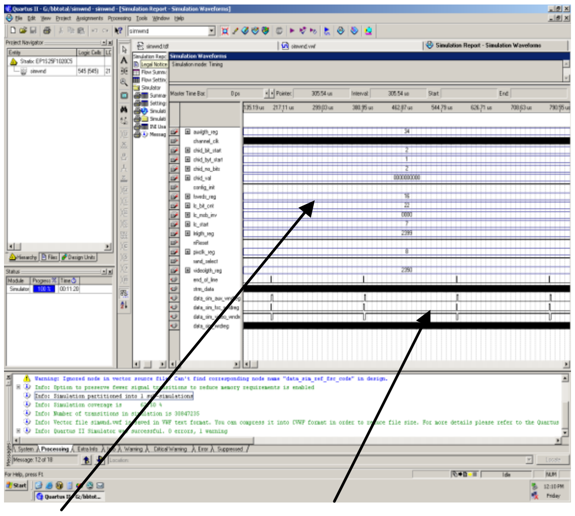

Data Flow
Real time Data Acquisition Frame Synchronization and Derandomization done in
Interface to Host for logging data

Fig8. Data Flow diagram of Hardware Derandomization

By implementing the Derandomization process in FPGA, the process takes place in real time, the computational overhead on the host processor is reduced, because the data to the host need not go through the derandomization process in software and thus total performance of the system is scaled up.

VII. SIMULATION RESULT



```
Warning: Ignored node in vector source file. Can't find corresponding node name "data_sim_ref_fsc_code" in design.
Info: Option to preserve fewer signal transitions to reduce memory requirements is enabled
Info: Simulation partitioned into 1 sub-simulations
Info: Simulation coverage is      62.50 %
Info: Number of transitions in simulation is 30847235
Info: Vector file simwnd.vwf is saved in VWF text format. You can compress it into CVWF format in order to reduce file size. For more details please refer to the Quartus
Info: Quartus II Simulator was successful. 0 errors, 1 warning
```

Configuration Parameters **Derandomized output**

Fig 9. Configuration Parameters and Derandomized output.

VIII. CONCLUSION

In this paper we have presented an FPGA- based reconfigurable coprocessor. The proposed coprocessor was fully tested on Stratix FPGA based PCI –X card. Simulation results have been given. It is found to be very effective in implementing logic modules in FPGA. Further work includes the extension of co processor functions for decryption, decompression etc.

REFERENCES

[1] R.Pradeep, S.Vinay, Sanjay Burman, Dr. V.Kamakoti (2005). FPGA based Agile Algorithm-On-Demand co-processor.1530-1591/05 IEEE. Proceedings of the Design, Automation and Testin Europe Conference and Exhibition (DATE'05).

[2] Christian Mandl, Adolfo Fucci, Eugen Brenner 1997. A fast FPGA based Coprocessor supporting hard Teal-Time Search. 1089-6503/97 1997 IEEE.

[3] Altera Master Core IP Mt64 User Guide.

[4] Stratix Device Handbook, Volume 1 July 2005.

[5] Pillem Ramesh, Venkata Aravind Bezawada, K S N Vittal, Dr. Fazal Noorbasha (2012)." Design of 64-bit Peripheral Component Interconnect Bus at 66MHz".

[6] Ali Azarian, Mahmood Ahmadi (2009). Reconfigurable Computing Architecture. 978-1-4244-4520-2/09 2009 IEEE.

[7] Mahendra Samarawickrama, Ranga Rodriga and Ajith Pasqual. HLS Approach in Designing FPGA based Custom coprocessor for Image processing. 978-1-4244-8551-2/10 2010 IEEE.

[8] Haber .J (2003)." Using a commercial IP core in space flight avionics. Lessons learned".

[9] Memory System Design from Altera Corporation February 2010.

AUTHORS

First Author –

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.

Second Author – 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)