FPGA Cluster Based High Throughput Architecture for Cryptography and Cryptanalysis

Diksha Moolchandani
IIITDM Jabalpur
Jabalpur, M.P. - 482005
India
dikshamoolchandani@live.com

Jayant Sharma
IIITDM Jabalpur
Jabalpur, M.P. - 482005
India
jayantsharma1@live.com

Abhishek Bajpai
BARC Mumbai
Anushakti Nagar, Mumbai - 400094
India
abbajpai@barc.gov.in

Saket Saurav
IIITDM Jabalpur
Jabalpur, M.P. - 482005
India
saket@iiitdmj.ac.in

ABSTRACT
This paper presents a highly parallel architecture for computationally intensive cryptanalysis applications using a cluster of FPGAs. This design proposes a generic and an expandable hardware network with high speed communication protocols based on Multi-Gigabit Transceivers and PCI Express bus achieving a maximum data rate of 2.5Gbps per FPGA. Main problem in such a design is management and transfer of bulk data while keeping the application interface simple. The proposed layered architecture provides a seamless data communication between Host PC and crypto core in an abstract manner. Performance much higher than cluster of CPUs and GPUs can be achieved at lower costs due to the flexibility and power efficiency provided by FPGA.

CCS Concepts
• *Computing methodologies~ Parallel computing methodologies*

Keywords
FPGA; cryptanalysis; PCI Express; MGT; SDRAM; SERDES; Cluster

1. INTRODUCTION
In recent years there has been an increase in requirement of data and network security due to which there is always a demand for more computation power and high speed communication. Cryptanalysis techniques like Distributed TMTO attack require as many instances of complete algorithm running simultaneously as possible based on the resources and time available. Also computations involved in cryptographic algorithms are intensive because of the large amount of logical and arithmetic operations required, thus the entire computation needs to be divided into elementary functions thereby parallelizing the computation and reducing its time. FPGA is capable to replace CPU when dealing with applications involving such a parallelism due to its lower implementation cost and higher time-performance ratio.

Building a cluster of devices is a general approach for increasing performance of a system. Disadvantages of this approach are low communication bandwidth and high cost. Modern approach is to offload CPU while maintaining a hardware-software tradeoff. This requires that devices share resources with a high data throughput. Enhanced version of COPACOBANA [1] utilizes 16 plug-in modules each hosting 8 FPGAs supporting direct binary addressing. It uses Ethernet interface to communicate with the Host but communication on plug-in DIMM module is via parallel buses. FPGA specific modules pose many difficulties like routing, EMI problems of parallel buses and scalability of design.

We propose a generic and flexible architecture involving a cluster of FPGAs arranged in the form of a ring network with each FPGA as its node. Size of the ring is expandable suiting the needs of application and high speed serial I/Os are used for communication. Host PC controls the operation of design and communicates with single FPGA using high speed PCI Express (PCIe) bus. This FPGA device acts as source node for the ring network and all data to and from Host passes through it. PCIe bus has been used to avail the maximum bandwidth and to eliminate the limitations of parallel bus architecture. Different devices communicate in ring via Multi-Gigabit Transceivers (MGTs) using a pair of SMA cables. MGT is a SERDES[1] used for high speed data communications with limited resources as they cost less than parallel interfaces with equivalent data throughput. So, MGTs are preferred because of the low cost involved and benefits provided by serial differential signaling over parallel buses.

In this design each FPGA contains several functions which operate together independent of the others. The number of such functions to be implemented is decided by the FPGA logic cells consumed by each function. Each function is allocated a memory space in the DDR3 SDRAM as a First In First Out (FIFO). Data packet containing the address of the destination function and device is sent by Host to the source node of ring using PCIe link. In the ring, packet keeps on traversing the nodes unmodified until it reaches the destination node where it is stored in FIFO for the targeted function. A DMA Controller on each device handles all memory access operations. Data to be received by Host is sent in the same manner across ring to the source node.

This architecture can be used for a vast number of applications such as design and testing the computational security of algorithms like implementation of high throughput AES core and

[1] Serializer/Deserializer (SERDES) core consists of Serial In Parallel Out (SIPO) and Parallel In Serial Out (PISO). It can be provided as a chip or an IP core and is used for high speed communication with serial/differential interface.

Parallel Pollard Rho's Algorithm for Elliptic Curve Discrete Logarithm Problem as presented in [2].

2. ARCHITECTURE

The design follows a layered architecture abstracting application interface from the lower layers of network design, though the number of layers is kept minimum to reduce complexity. This enables us to independently update and maintain different components like user design, application access and communication media (UART, Gigabit Ethernet or GTX Transceivers). Application Layer is controlled at Host level which determines the requirement and then selects and initializes different functions from available options. Data Link Layer deals with data transfer structure and protocols using Intellectual Properties (IP) cores and physical components. The Network Layer contains the logic and data flow paths developed for the functioning of design. User Design containing user functions is a part of Application layer implemented on FPGAs. Figure 1 shows the overview of layered architecture along with directional flow of data and approximation of resources used by different cores. So, we will now describe the layers included in design and their internal structure in detail.

2.1 Application Layer

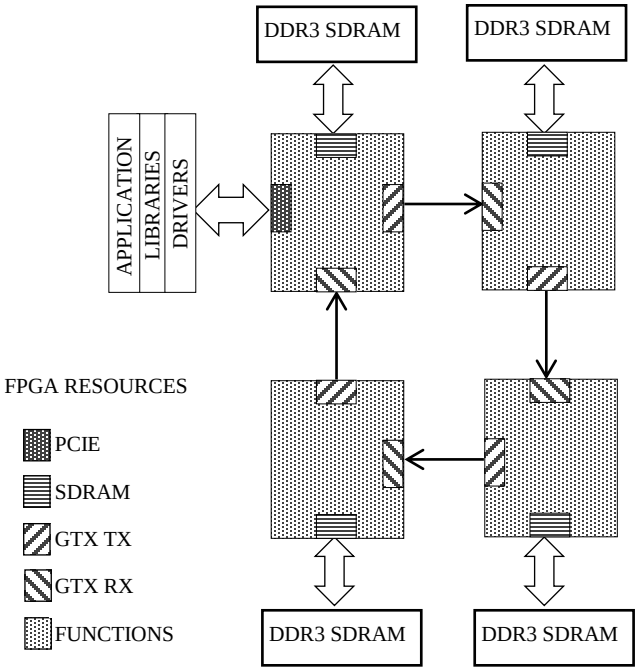

Figure 1. Architecture of Complete Network with four FPGAs showing device utilization.

Application Layer is maintained at the Host PC which controls the selection of functions to be required by application by sending structured packets. Application communicates with a Python wrapper code which provides read/write access into the device memory space called Base Address Registers[2] (BARs). BARs represent memory spaces as seen by the host system to communicate with the PCIe device. Host can access configuration registers and BARs with the help of a generic open-source Linux pciDriver and MPRACE library provided by [3] available at [4]. The device driver accessed by a C abstraction layer maps PCIe BARs to Host's memory space. PCIe endpoint device holds two BARs - BAR0 is used as configuration space and BAR1 is 32KB and 64 bit wide space for user design to read and write. The data needs to be transferred to function performing desired operation on the targeted FPGA. Packets are used to transfer data to every function on each FPGA with maximum payload of 1024 bytes. The whole packet is written into 256 bit wide dual-port BRAM of size 8KB which is assigned to a part of BAR1. Status Registers holding status of all functions are assigned to the remaining part of BAR1.

[2] In a network interface RAM memory buffer of some I/O device must be mapped into system's memory space. This RAM is allotted to Base Address Registers, also called BARs. BARs provide mechanism that allows configuration software to determine the memory mapped I/O space that a device requires.

2.2 Data Link Layer

Data Link Layer provides transparent network services by checking end-to-end validity of data being transferred and error handling. It contains FPGA logic needed to develop interface between user design and physical links connecting Host CPU to FPGA via PCIe and FPGA to another FPGA via Multi-Gigabit Transceivers (MGT) using SMA cables. PCIe and MGT IP Cores are used to provide the data link functions like CRC and data integrity check. The 4 lane Gen1 PCIe gives a data rate of 2.5Gbps. Aurora protocol is used for implementing MGTs on FPGA. Maximum line rate of 3.125Gbps can be achieved when data is transmitted between two devices over SMA cables. A ring network topology is used to connect all the devices with unidirectional flow of data across the ring. The design provides flexibility to expand ring to any number of devices suiting the requirements of design.

Table 1. Packet structure with contents in the order of transmitting.

Packet	Size (bytes)	Description
PKT START	1	Start byte for packet
DEVICE ID	1	Device ID of targeted FPGA
UFID	2	Unique Function ID of targeted function on FPGA
COMMAND	1	Control Commands like READ/WRITE/RESET etc.
LENGTH	2	Total number(*data_sz*) of data bytes
ADDRESS	2	Starting address for memory
DATA	*data_sz*	Data payload of max. 1024 bytes
PKT END	1	End byte of packet

2.3 Network Layer

Network Layer provides routing infrastructure and packets processing. Host PC is connected only to first FPGA device Board enabled with PCI Express physical link which connects the Host to other devices. Data in the form of packets is transferred from Host PC and written to in-BRAM on the source node. Packet structure includes data to be transferred and identification information of the targeted function as shown in Table 1. Device ID of every packet stored in in-BRAM is checked to find out its destination FPGA. If the packet belongs to this device, it is written to the corresponding FIFO in DDR3 SDRAM, otherwise it is sent to GTX Core to be transmitted to next device in ring. A DMA Con-

troller on this device handles data from both PCIe and GTX Receiver and routes it to either SDRAM or GTX Transmitter based on destination. On other devices the packet is either written to SDRAM or sent to next device and similar procedure follows up on next device. Data path to the packets is provided using components like local buffers and DMA Controller as shown in Figure 2.

2.3.1 Local Buffers

Data from functions needs to be stored in local buffers as SDRAM can be accessed by only one function through DMA. The buffer size of 8KB is chosen as it can be easily implemented efficiently by using two BRAMs. If all functions require such buffers, a total of 234 functions can be implemented per Virtex-6 FPGA. But the number of functions is limited by logic used by single core of algorithm function and it is generally less than 234.

2.3.2 DMA Controller

To avoid data traffic conflict while accessing the SDRAM, a DMA Controller is implemented which controls access to memory bus based on requests using a priority scheme. PCIe request from Host is given the highest priority followed by SMA and then other functions. FPGA functions are of same priority and are handled in round robin manner. A masked priority arbiter grants DMA to a function based on the current request and mask vector. The mask vector is generated by the grant of previous DMA operations, thus masking devices which have been given grant earlier.

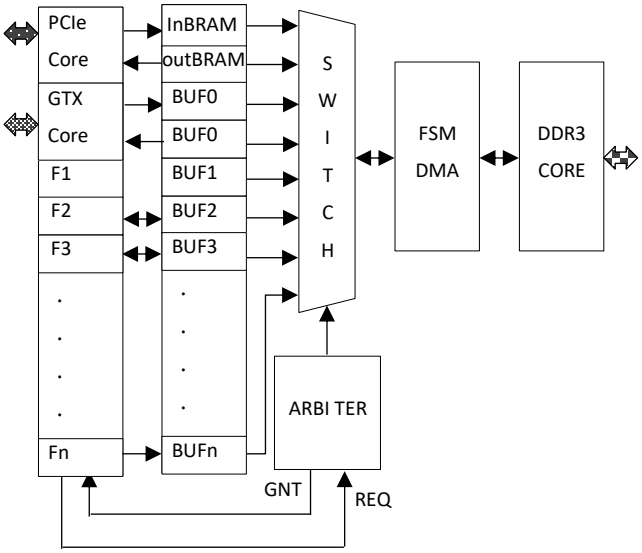

Figure 2. Components of Infrastructure Layer.

Furthermore, the design of the DMA Controller consists of Finite State Machines that are used to access data from the local buffer of the device with DMA grant and then redirect it to the DDR3 Core which exchanges data with SDRAM. Also, this FSM implements SDRAM as a FIFO buffer for each device so that it cannot be read or write when buffer is empty or full respectively. Address space of the DDR3 memory comprises of RANK[26], ROW[25:13], BANK[12:10] and COL[9:0], although these depend on the memory being used. But in our design a single function is given single row and corresponding fully utilized 1024 columns (COL[9:0]) for providing 8KB buffer memory. Thus, 256 rows (ROW[20:13]) are required for 256 devices. A total of maximum 8192 functions can be implemented if required. There are total 8 banks of which one is sufficient for single FPGA. So, a single DDR3 memory can be shared between 8 FPGA devices if a multi-device board is used.

2.3.3 Status Registers

Status registers are allotted to each function on a FPGA device to which the function writes back after completion of operation. Status registers of functions of all devices are stored on the source node device. This reduces the overhead of sending readSTATUS command to each device separately. Status registers are mapped to a 16KB space of BAR1 which can be polled by Host at any time.

2.3.4 Function Descriptor

A Descriptor organizes various Function parameters as a Doubly Linked List structure *FuncList.* It lists all devices and their functions as a descriptor space on source node and maps this space to BAR1. New devices can be added to the ring just by updating this list. Host gets information about available devices and their functions by reading BAR1.

```
struct FuncList {
   int UFID;
   int FID;
   int *Status;
   int *Command;
   struct FuncList *next;
   struct FuncList *prev;
}  *device;
```

2.4 User Design

User Design contains functions to be implemented on each device according to the application. Each function on FPGAs is defined by a Device ID (DID), Unique Function ID (UFID), Function ID (FID) and status registers. The number of function cores that can be implemented practically depends on three major factors: SDRAM size, FPGA size and on-chip local memory requirement of each function. Following procedure is used to calculate the maximum number of functions:

i. Assuming that memory used is a standard DDR3 SDRAM SO-DIMM[3] of RANK 'r' and each function requires memory less than 8KB, maximum number of functions implemented: $n_1 = (8192 \times r)$.

ii. Assuming that the FPGA device used is Virtex-6[4] and each function requires 'x' KB memory for its local buffer, number of functions implemented: $n_2 = (2048 \div x)$.

iii. Logic Cells for a particular device determine number of cores (say, n_3) that can be implemented. For small applications like RC4, more than 1024 cores can be easily implemented while for large applications like AES-256, it is only about 25 cores.

[3] Total size of DDR3 SDRAM SO-DIMM memory module is calculated as: (RANK × 4chips × 8banks/chip × 8192rows/bank × 1024columns/row × 16datawidth/column).

[4] Xilinx Virtex-6 XC6VLX240T FPGA contains 416 36Kb BRAMs memory and 832 18Kb BRAMs i.e. 3744KB. Of this we are using 2048KB memory only.

Thus, the number of functions on a device is given by $\min\{ n_1, n_2, n_3 \}$, and is usually determined by n_2 for smaller cores and by n_3 for larger cores.

3. RESULTS

The proposed architecture was implemented and tested using four ML-605 Virtex-6 FPGA Boards which were featured with PCIe link, SMA ports and DDR3 SDRAM. Timing delays due to data transfer via PCIe bus and GTX Transceivers were calculated on-chip by transaction of test packets. For a data size of 32 KB, maximum throughput measured is about 867MBps for memory write operation which is close to the theoretical throughput of 1000MBps for simplex mode while 750MBps is a constantly appearing rate. For memory read operation, 516MBps throughput is achievable using DMA access. Data transfer using GTX Transceivers and SMA Cables occurs at line rate of 3.125Gbps with single line being used. Total time taken for reading 8KB data from in-BRAM, sending it to next FPGA over SMA and writing in a local buffer is about 30us at 200MHz clock frequency. Parallel applications like cryptanalysis need to initialize all the functions together giving some data and commands. The total time taken for the process is minimum when the order of sending data is last to first device with a delay of 30us between two consecutive devices. This procedure takes a total time of $30n$ us for sending data to n devices. For a loop of 100 devices and 256 functions per device, all 25600 functions can be initialized in 0.8s which is negligible as compared to the total time taken by cryptanalysis.

Table 2. Resource Utilization of all cores on a single FPGA

Cores	Slices	LUTs	Registers	Bram 36Kb
PCIe Core	3768	10269	12556	20
GTX IP Core	4	3	3	0
DDR IP Core	2291	3551	5322	0
DMA with 1 Function	2180	3583	5302	2
DMA with 10 Functions	2188	3592	5302	20

We used 512MB DDR3 SDRAM which permits upto 8192 functions/device. Buffer size for each function is 8KB so, 256 functions can be implemented while leaving half the memory (18Kb BRAMs) still unused. The memory not used for buffers is utilized in other cores of design like PCIe core, GTX Aurora core and user design. The resources utilized by source node device is more than that used by other devices due to overhead of Status registers and PCIe core. The summary of logic and memory utilized by each core is given in Table 2 thus showing the resources left for user design logic i.e. application functions. The overhead incurred by PCIe core is also shown.

4. CONCLUSION

In this contribution we present a scalable parallel hardware architecture for cryptanalysis applications. This architecture is suitable for many applications of different sizes and complexity. The flexibility provided enables it to be used with various FPGAs supporting MGTs or Gigabit Ethernet. User can configure various parameters like device size, memory size and number of devices according to the speed and cost requirements. This system can also be implemented with some changes in Data Link Layer, on Boards that contain multiple FPGAs with single parallel bus connecting them. A common SDRAM can be shared between many devices provided there is physical link between DMA Controller and all devices. We plan to use it for hardware implementation of high throughput AES-256 and NIST Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications [5].

5. ACKNOWLEDGMENTS

The work presented in this paper was jointly supported by Bhabha Atomic Research Centre (BARC), Mumbai and Indian Institute of Information Technology, Design and Manufacturing, (IIITDM) Jabalpur. This work has been carried out by authors as students at IIITDM Jabalpur and working as trainees at BARC during the execution of the work. We express our thanks to the scientists and staff at Computer Division, BARC for their support in providing the inputs and hardware used for the research.

6. REFERENCES

[1] T. Gueneysu, C. Paar, G. Pfeiffer, and M. Schimmler, "Enhancing COPACOBANA for Advanced Applications in Cryptography and Cryptanalysis", Sep. 2008, FPL, Heidelberg, Germany.

[2] CUDA based implementation of parallelized Pollard's Rho algorithm for ECDLP. CHINNICI et al. FINAL WORKSHOP OF GRID PROJECTS,"PON RICERCA 2000-2006, AVVISO 1575".

[3] The MPRACE framework: An open source stack for communication with custom FPGA-based accelerators. Marcus, G. ; Wenxue Gao ; Kugel, A. ; Manner, R. Programmable Logic (SPL), 2011 VII Southern Conference on DOI:10.1109/SPL.2011.5782641 Publication Year: 2011 , Page(s): 155 – 160.

[4] G. M ARCUS, 2011; MPRACE Software Library, Institut für Technische Informatik Heidelberg, http://li5.ziti.uni-heidelberg.de/mprace/

[5] "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," NIST Special Publication 800-22, Revision 1a.