FPGA Cluster based high performance Cryptanalysis framework

Abhishek Bajpai
Computer Division,
Bhabha Atomic Research Centre,
Mumbai, India
Email: abbajpai@barc.gov.in

S V Kulgod
Computer Division,
Bhabha Atomic Research Centre,
Mumbai, India
Email: svkulgod@barc.gov.in

***Abstract*—In this paper a 'FPGA cluster' based framework for high performance Cryptanalysis has been proposed. The framework abstracts underlying networked FPGA cluster into a unified acceleration resource. It does so by implementing requested amount of computation kernels (crypto-graphic modules) and managing efficient distribution of the network bandwidth between the inter-FPGA and intra-FPGA computation kernels. Further agile methodology for developing such networked computation kernels with use of a high level language (Python) based HDL library and seamless integration with a user space crypt analysis application have been discussed. 40-bit partial key attack over AES256 has been demonstrated as a capability demonstration. Performance higher than clustered CPUs and GPUs at lower costs and power is reported.**

***Keywords*-FPGA; cryptanalysis; cluster; framework; high performance computing;**

I. Introduction

Cryptanalysis is a study of characterising a cryptographic algorithm for its weaknesses and using these weaknesses to decipher the ciphertext without knowing the secret key. Sometimes the weakness is not in the cryptographic algorithm itself, but rather in the implementation or the protocol, that makes cryptanalysis successful. An attacker may have very limited goals as well, like gaining just small part of the information or just distinguishing the algorithm itself. Characterisation of a cryptographic algorithm is generally based on mathematical and statistical observations. Often Statistical observation based characteristic discovery is used as a primary tool in the cryptanalysis followed by mathematical observations and vice versa. For example, Roos biases for the first few bytes of RC4 were first observed by Andrew Roos and proved by Goutam Paul et. al. [1] Later Nadhem AlFardan et. al. [2] statistically searched similar biases for first 256 bytes of the keystream. Present day cryptography heavily relies on such high-throughput statistical techniques for algorithm characterisation. Most of these characterisation techniques require running parallel instances of simpler algorithms over a large amount of data for a large amount of time. Such characterisation techniques include:

- High performance online Statistical test suit for randomness test
- Bias discovery
- Linear Crypt-analysis
- Differential Crypt-analysis
- TMTO (Time Memory Trade Off), TMDTO (Time Memory Data Trade Off) based attacks
- Collision Attacks

A. High-Performance Computing (HPC) systems

For HPC systems, clustered devices is a general approach to begin with. Though clustering results in low communication bandwidth and high cost as compared to integrated systems. Yet it is the fastest way to achieve the HPC setup. Further, Heterogeneous High-Performance Computing (HHPC) is about offloading CPU with an another computing component (accelerator) while maintaining a hardware-software trade-off.

GPUs, because of their high count computing elements and fine-grained architecture, staged as a good candidate for HHPC accelerator. Though they are limited with SIMD (Single instruction, multiple data) instructions set. Thus, supports simultaneous (parallel) computations, but only a single process (instruction) at a given moment. These architectures exploit data level parallelism, but not concurrency.

FPGAs are extremely fine grained and have massively interconnected architecture. It's basic computing element (logic block) comprised of few small bit sized function generators(LUTs). Thus, supports different simultaneous computations with concurrency. These properties allow implementing massively parallel pipelined cryptanalysis stages. Our initial work on AES implementation [3] reports significant advantage over processors based implementations. The main drawback of this architecture is it's difficult and complex development cycle.

B. Related Work

In 2005 Chang, Wawrzynek et al. proposed a reusable, modular, and scalable framework for high-end reconfigurable computers, the BEE2 project [4]. They also proposed Mathworks Simulink based programming model. The system was designed with tightly bounded latencies for the in-system data transfers, BEE2 was well suited for real-time applications requiring high integer or fixed-point computational throughput. The BEE2 system was observed

to provide over 10 times more computing throughput than a DSP-based system with similar power consumption and cost, and over 100 times that of a microprocessor-based system.

EPCC (Edinburgh Parallel Computing Centre) a founding member of the FPGA High-Performance Computing Alliance (FHPCA), in 2007 developed a general-purpose 64-FPGA Supercomputer "Maxwell" [5]. The system comprised of FPGAs interconnected in a two-dimensional torus structure based on IBM Intel Xeon blades. Bacause of fast FPGA-FPGA links it was highly suitable for nearest-neighbour communication patterns.

In 2006, Kumar, Paar et al., [6] proposed COPACOBANA "The Cost-Optimized Parallel Code Breaker", based on FPGA clusters and optimised for cryptanalysis applications. The system comprised of DIMM modules, containing 6 Xilinx Spartan-3-1000 FPGAs, connected over a shared 64-bit data bus on a DIMM backplane. In a 2007 implementation proposed by Gneysu, Kasper et al. [7], the system running at 136MHz could search a full 56-bit DES key space in 12.8 days. FPGA-specific modules pose many difficulties like routeing, EMI problems of parallel buses and scalability of design.

Cube, [8] a massively-parallel FPGA-based platform was presented in 2009 by Mencer, Hung et al. The machine comprised of 512 FPGA devices where 8x8 FPGAs arranged on the baseboard and 8 similar boards stacked together to form an 8x8x8 3D matrix (cube) structure. Fast inter-FPGA communication network and a flexible programming scheme resulted in a low power, high density and scalable design supporting various large-scale parallel applications. The design was demonstrated with an RC4 key search engine implementation.

During same time A heterogeneous computer cluster named Axel was proposed by Tsoi, Luk et al. [9]. Axel was a collection of nodes including several accelerators types such as FPGAs and GPUs (Graphics Processing Units). They introduced a novel Map-Reduce framework which maintains spatial and temporal locality in computation, through different types of processing elements and communication channels. The Axel system is alleged to be the first one which demonstrates FPGAs, GPUs and CPUs running collaboratively for N-body simulation. They claimed performance improvement from 4.4 times to 22.7 times using their approach.

Recently in 2016 Zhang, Wu et al. [10] presented a deeply pipelined multi-FPGA Cluster for Energy-Efficient CNN (Convolution Neural Networks) Implementation. They also proposed a dynamic programming algorithm to map the CNN computing layers efficiently to different FPGA boards connected over high-speed serial links. The experimental results on AlexNet and VGG-16 showed that the prototype can achieve up to 21X and 2X energy efficiency compared to optimised multi-core CPU and GPU implementations, respectively.

key contribution: In this work a generic and flexible framework have been proposed which supports FPGA clusters in various network topologies. The size of a network is expandable suiting the needs of applications. The framework utilises high-speed interfaces and specifies basic building blocks for interconnects. Framework supports various network configurations and specifies packet structures and packet routeing protocols through the network. Further the framework was developed over a high-level language (python) in form of a library "FHPClib". As it is based on Python, "FHPClib" enables agile development cycles reducing development time significantly.

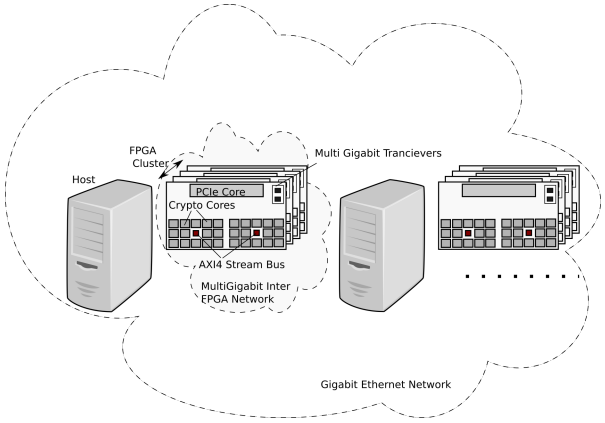

Figure 1. Cluster

II. Architecture

In this design, a couple of COTS (Component of the shelf) FPGA board are connected to the host over PCIe and they are also interconnected over a Multi-Gigabit Transceiver Interface (MGT) to form a local FPGA network (see figure:1). Moreover, multiple such hosts are connected in a network to form a cluster.

Further, each FPGA contains several crypto kernels (crypto compute modules) which operate independently. The number of such kernels is decided by the FPGA resources consumed by each kernel.

The design follows a layered architecture abstracting underlying network from the user level cryptanalysis application and crypto kernels (compute modules) implemented over FPGA cluster (see figure:2). The Data transfer takes place in the form of packets between kernel to kernel and host to kernel along these layers(see figure:3). These packets route through the network via packet switches and AXI4 buses arranged in different network topologies. The architecture can be divided into two major parts from the implementation point of view.

- Host (Software)
- FPGA Cluster (Hardware Core)

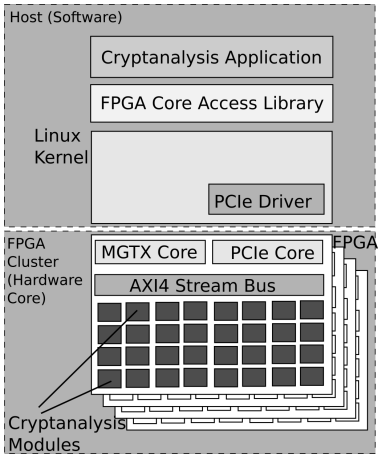

Figure 2. Architecture

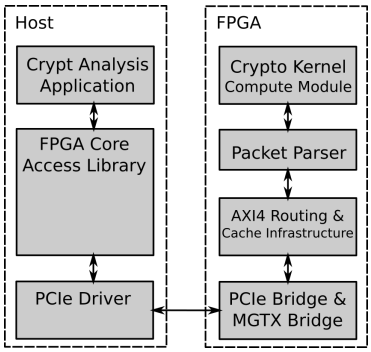

Figure 3. Data Flow

A. Host (Software)

The host is responsible for control and communication of job packets to the FPGA cluster and also for communication between hosts. Host components comprised of three main components.

1) User Level Crypt-analysis Application: User Level Crypt-analysis Application is basically a user interface from where a user can set various cryptanalysis parameters. It further divides the main problem into finer multiple parallel jobs and passes them to the FPGA Core Access Library. It receives asynchronous responses (job results) from the library and compiles them in a user-friendly diagrams/compiled results. A high-level language (Python) has been used for faster development.

2) FPGA Core Access Library: FPGA Core Access Library is majorly responsible for providing abstract FPGA core interface to user Application. It functionality includes

- Initialises FPGA clusters, underlying network and Crypto Kernels
- Maintains a database of the methods provided by each crypto kernel with their physical address
- Arrange asynchronous jobs from the user application
- Allocation of job numbers and packet encapsulation of jobs with network headers, based on available free crypto kernels
- Forward job packets to the PCIe FIFO (First In First Out) bridge driver
- Mapping received response packets from the PCIe driver with the corresponding job number
- Forwarding results/response to the user application via callback interface for further result/data compilation

3) PCIe FIFO bridge driver: The PCIe driver has been developed with a FIFO bridge configuration. Development was based on Marcus et. al. [11] work. It is developed to interface FPGA PCIe Packet FIFO Bridge Core. The driver basically maps job packets memory buffers with the FPGA's internal FIFOs. Further, a driver is also responsible for resetting, configuring and status polling of various parameters of the FPGA Cluster.

B. FPGA Cluster (Hardware Core)

FPGA Cluster architecture may further be divided into three major design elements.

- Interfaces
- Network Elements
- Crypto Kernels

C. Interfaces

Various high speed interfaces have been developed in order to achieve high throughput communication between Host-FPGA and FPGA-FPGA networks.

1) PCIe Packet FIFO Bridge: PCIe FIFO Bridge core is developed as a FIFO cache to forward job packets from the host machine on a first in first out basis. For this functionality PCIe's TLP (Transaction Layer Packets) DMA (Direct Memory Access) core is developed which maps host memory blocks with its internal FIFO dynamically. Further control/status logic is also developed to configure FPGA Cluster and user application memory FIFOs. DMA (Direct Memory Access) relieves the host processor by directly fetching job packets and pushing results directly to the user memory. It is developed in order to achieve high throughput communication between crypt-analysis application running on the host and the crypto kernel (compute module) on the FPGA. With the present development, we have achieved 2.5 Gbps throughput over 4 lane PCIe interface.

2) MGT(Multi Gigabit Transceiver) Packet FIFO Bridge: MGT(Multi-Gigabit Transceiver) is a Serializer/Deserializer (SERDES) consists of Serial In Parallel Out (SIPO) and Parallel In Serial Out (PISO). It can be available as a chip or an IP core and is used for high-speed communication with serial/differential interface. As compared to parallel interfaces, routeing of SERDES differential signals is less complex for equivalent data throughput. Thus, MGT is preferred choice for inter-FPGA high-speed data communications.

MGT Packet FIFO Bridge core is developed to do inter-FPGA packets transactions. With the present development, we have achieved 2.5 Gbps throughput.

3) DDR3 (Double data rate type three SDRAM) Packet FIFO Cache: DDR3 Packet FIFO Cache has been developed to cache data packets locally on the FPGA board. This helps in achieving bulk transfers by detaching high-speed paths and the crypto kernels by behaving as an in transit buffer. Because of this, high-speed paths don't get overwhelmed as crypto kernels internal buffers get full. Thus, high-speed paths are always ready to accept packets.

D. Intra-FPGA Network Elements

Cryptanalysis computations are subdivided and formatted into job packets and results of these intensive job computations are not instant. Thus, jobs are processed in the framework asynchronously. These job packets flow through the network via various network elements as streams and processed on first come first serve basis.

1) Modified AXI4 stream bus [12]: Intra-FPGA network is designed to be based on AXI4 stream bus. AXI4 stream is used for high-speed streaming data and is the part of ARM Advanced Microcontroller Bus Architecture (AMBA) family. It is an open-standard on-chip interconnect specification used in system-on-a-chip (SoC) designs for the connection and management of the functional blocks.

As AXI4 stream bus is having separate read and write lanes, a master can simultaneously read and write streams to/from slaves achieving higher throughput. Further slaves (crypto kernels) are arranged down the stream bus in a daisy-chained fashion(see figure:9). Pipelining is being implemented at every kernel connection in order to reduce path delays and achieve timing constraints.

The bus is configured in a single master and multiple slave mode. Apart from other generic signals [13] (see table:I) 'tuser' of the read lane is customised for passing access tokens among slaves by a bus master. Based on this access token, slaves can send result packets back to the host or further, forward job packets to another kernel.

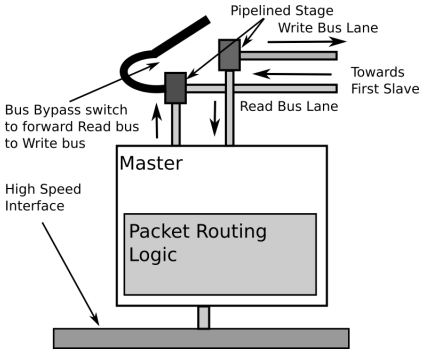

Figure 4. AXI4 Master/Forwarder

Table I
AXI4 STREAM BUS SIGNALS

Signal Name	Size	Direction
Write Bus		
tvalid	1 bit	Master → Slave
tready	1 bit	Slave → Master
tdata	64 bit	Master → Slave
tkeep	8 bit	Master → Slave
tlast	1 bit	Master → Slave
tid	4 bit	Master → Slave
tdest	4 bit	Master → Slave
tuser	4 bit	Master → Slave
Read Bus		
tvalid	1 bit	Slave → Master
tready	1 bit	Master → Slave
tdata	64 bit	Slave → Master
tkeep	8 bit	Slave → Master
tlast	1 bit	Slave → Master
tid	4 bit	Slave → Master
tdest	4 bit	Slave → Master
tuser	4 bit	Master → Slave

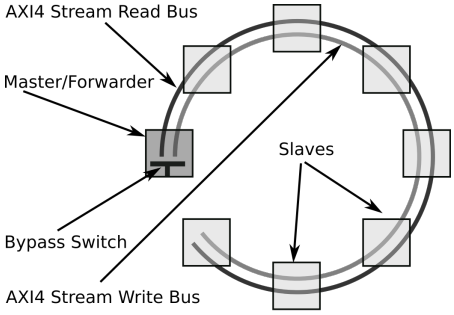

Figure 5. Modified AXI4 Ring

2) AXI4 bus Master/ forwarder: AXI4 bus Master/forwarder controls job packet movement over the AXI4 bus. Due to separate read and write buses Inter-slave communication is not possible by design. but with few tweaks, inter-slave communication and local network broadcasting are also made possible. For inter-slave communication, the bus master is developed with an extra capability of bypass switch between read and write bus lanes (see figure:4). When master sense that a slave wants to communicates with the other slave in the local network its sets itself into a bypass mode. Bypass mode forward read bus lane packets coming from slaves to the write bus lane so that it gets routed to the destined slave. The only drawback is, while in this mode, throughput drops to half as both the buses gets used while single packet transactions. Typically bus masters are connected with a high-speed interface like PCIe bridge or MGT bridge but it can also be connected as slave on the another AXI4 bus just to extend the network (see figure:6). Such extensions don't overload the root bus while local inter-

kernel communications.

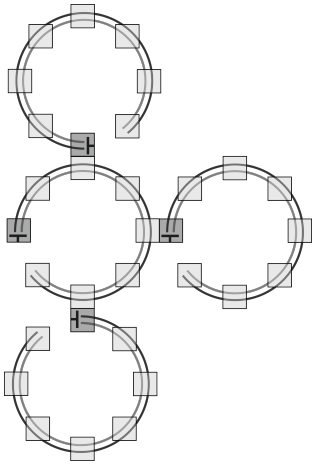

Figure 6. Modified AXI4 Networked Rings

3) AXI4 Slaves: AXI4 slaves are supposed to receive job packets from the master based on their unique identifier, they can also receive broadcast packets if 'tdest' is set to broadcast identifier. They are daisy chained on an AXI4 bus (see figure:7). Since a packet transaction, either in read lane or write lane blocks that particular lane, further slave-slave packet transaction blocks the whole lane, there is an upper cap on slave counts on an AXI4 bus. A Large number of slaves may introduce the communication bottlenecks because of the large throughput requirement. For an efficient slave to slave communication, one should place all the interactive kernels on the same local bus.

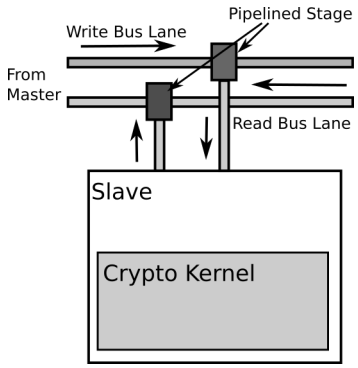

Figure 7. Pipe-lined Slave connector over AXI4 bus

E. Crypto Kernel and other basic modules

Crypto Kernel is the smallest function (method) specific compute module of a larger granular design. It is responsible for a simpler functionality which is needed to be computed in a highly parallel fashion. Apart from unified status/command get/set mechanisms, the framework also specifies method/function declaration request/response mechanism based on unique method/function identifiers. Each kernel is allocated and identified by a unique network address and unique method/function identifier based on its functionality.

1) Packet FIFO (First In First Out) Buffer: Packet FIFO (First In First Out) buffers are used as an endpoint receivers as in slaves and also as intermediate transit queues to cache the packets and forward it to the next hop as in bus masters and high-speed bridges. They are preferred to dual port ram buffers, as data flows in the form of packet streams in the network. FIFO also reduces the routeing cost as address lines are not required to be routed in contrast with ram buffer design. Further reducing the design complexity.

2) Packet Parser: Packet Parser is a terminating node in a network and abstracts kernel functionality. It exposes kernel functions through data structures. It parses packets from the network and extracts underlying data structure, that is meant to be arguments for the kernel. It initializes the kernel module with extracted arguments data and runs. After execution, it fetches the result. Further, It composes the result in a data structure, encapsulates it in a packet adding header information and send it back to the host/source.

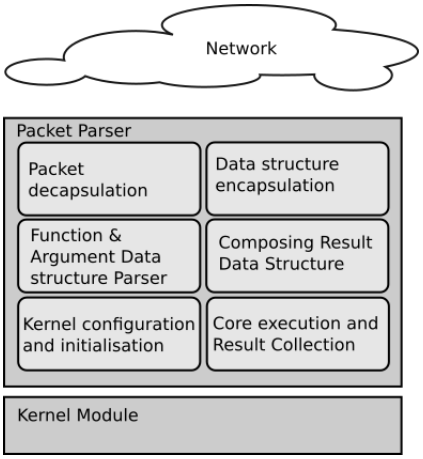

Figure 8. Packet Parser

F. Job Packet

Data communication between the kernel pairs and host-kernel happens over a networked architecture in the form of packets. These packets get generated in the application layer where a cryptanalysis problem is subdivided into parallel jobs with distinct methods and data arguments. Thus, individual jobs get encapsulate into an application layer packets and header contains compute methods, allocated kernel id, job identifier (packet id) and various kernel configuration parameters.

Further, in order to propagate over the complex network, packets are appended with the network header containing

a length of packets, a destination identifier and a source identifier.

In order to synchronise boundaries of each packet over AXI4 bus and High-Speed bridges, one or more packets are encapsulated in the data link layer packet. Data link header presently comprised of a start of packet (see table:II).

Table II
PACKET STRUCTURE

	7	6	5	4	3	2	1	0
dlnk header	Start of Packet							
net. header	X	X	source		sink		length	
App. header	packetid		size		mode	nk	enc	Kid
Data	Key 0							
	Key 1							
	Key 2							
	Key 3							
	Data 0							
	Data 1							

III. HDL DEVELOPMENT

HDL(Hardware Descriptive Language) development is a lengthy and time-consuming process and also considered to be different from other programming languages. Because of this disconnect, new computer science advancements like unit test frameworks, agile development models etc are not directly applicable to HDL development. MyHDL [14](Python based HDL library) gives a new approach for Hardware description and verification at the higher level programming language. Such approach gives us an opportunity to bypass pre-synthesis functional analysis and also enable us to push (automate) synthesis, area planning, design implementation (translation, mapping, place and routeing), bit generation and programming to the back-end. Further making the whole design procedure fast and in-line with rapid development methodologies. Because of the higher level language (Python) and availability of rich library set, complex yet complete (real life scenario) test-benches can be created and verified for better test closures. Adding to it, we have developed 'FHPClib' (FPGA based High-Performance Computing Library) which encapsulates all the high-speed interfaces core IPs for Host to FPGA and FPGA to FPGA data communication and other basic building blocks IPs of the FPGA Cluster based high-performance Computing framework. Automatic creation of an intra-FPGA and an inter-FPGA network of kernel modules over a predefined cluster is carried out by the library. 'FHPClib' also have bypass implementations of all of its complex interfaces for faster functional simulations. Further, the library eases to write host HPC applications even before the HDL implementation by abstracting host FPGA interface drivers. Thus providing a seamless transition from simulation environment to the actual HPC set-up.

IV. FHPCLIB

'FHPClib' gives an high-level abstract interface which simplifies the network and node creation according to the framework. For example in the figure:9 a small network over AXI4 stream bus is created having one master node and five slave nodes. PCIe FIFO Bridge is connected to the master so that FPGA core access libraries can send/receive data packets over AXI4 bus network. One MGT FIFO Bridge is connected to a slave for inter-FPGA packet communication. Further, four AES256 crypto kernels are added to rest of slaves.

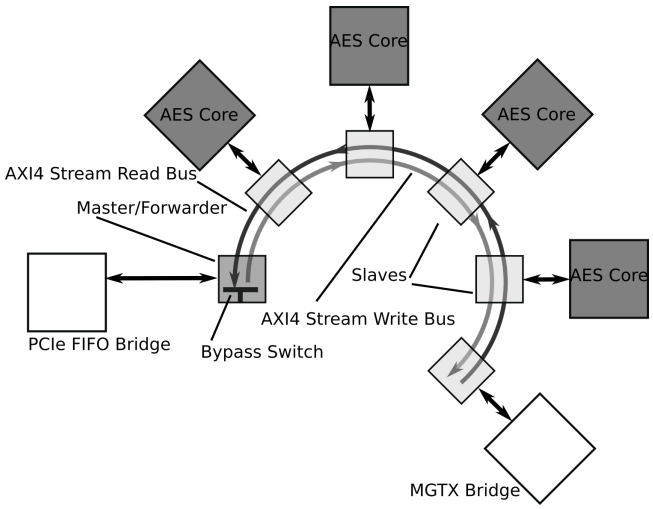

Figure 9. Network having AXI4 bus with PCIe FIFO bridge (master), 4 AES256 kernels (slaves) and 1 MGTX FIFO bridge(slave)

Listing:1 shows instantiation of such a network over 'FHPClib' library. The interesting part of Kernel instantiation is that it can simply happen in a for-loop. A lot of port declaration complexities, connection logics and cumbersome module instantiations are being abstracted by 'FHPClib'.

Listing 1. framework

```
Master_inst = PCIeFIFOMaster(clk,
                             aXI4bus[0],
                             rst,
                             DATA=DATA,
                             WORDS=WORD,
                             I_SIZE=I_SIZE,
                             D_SIZE=D_SIZE,
                             U_SIZE=U_SIZE,
                             MasterID=MasterId)
SlaveNum = 5
for i in range(SlaveNum-1):
    Slave_inst[i] = AXI4pktAES256Slave (clk,
                                 aXI4bus[i],
                                 aXI4bus[i+1],
                                 rst,
                                 DATA=DATA,
                                 WORDS=WORD,
                                 I_SIZE=I_SIZE,
                                 D_SIZE=D_SIZE,
                                 U_SIZE=U_SIZE,
                                 MasterID=MasterId,
```

```
                                SlaveID =i+2,
                                SIMULATION=True)

Slave_inst[-1] = AXI4MGTXBridgeSlave (clk,
                                aXI4bus[i],
                                aXI4bus[i+1],
                                rst,
                                DATA=DATA,
                                WORDS=WORD,
                                I_SIZE=I_SIZE,
                                D_SIZE=D_SIZE,
                                U_SIZE=U_SIZE,
                                MasterID=MasterId,
                                SlaveID =i+2,
                                SIMULATION=True)
```

Further, this implementation can be extended with multiple FPGA cluster (see fig:10) by arranging them in a ring network of High-speed Multi-Gigabit Transceiver lanes.

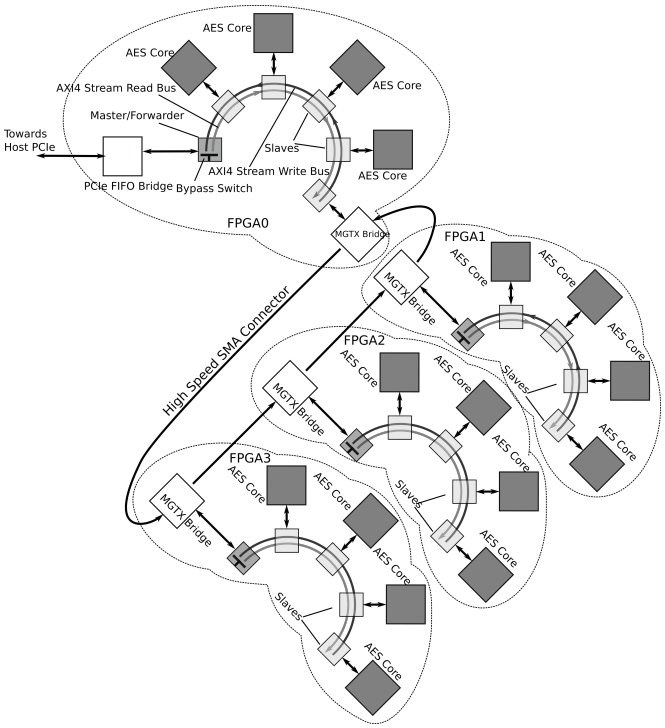

Figure 10. Cluster

A. comparison with OpenCL

OpenCL (Open Computing Language) [15] [16] is a parallel language specification aimed to provide portability across different platforms in a heterogeneous computing system. It consists of an API for coordinating computational and communicational tasks with the accelerators and a language based of C99 for describing the compute core at the finest granularity. In the context of an FPGA, it is a standard multi-core programming model that provides a higher-level layer of abstraction for FPGA's. The model has been studied and compared with 'FHPClib' development framework.

Table III
'FHPCLIB' VS 'OPENCL'

'FHPClib'	**'OpenCL'**
Based on Python	Based on C/C++ and OpenCL C
Uses Open Sourced MyHDL	Uses Close Sourced libraries
Code is translated (one to one mapping no optimization)	Design is inferred from a high level C code further optimized
Converted to readable Verilog/VHDL then rely on the vendor tools for generation of bit files	Converted to low-level unreadable RTL Design and further convert to bit file via OpenCL compiler
Tight control on HDL optimization	optimisation done via automated tools
Framework is based on packet switched network of kernels	Framework is based on memory mapped kernels
Inter Kernel communication is possible	Inter Kernel communication not possible
Supports only FPGA's	Supports any computing element (Processor, GPU, DSP, FPGA)

Table IV
AES-256 CORE (CRYPTO KERNEL) SPECIFICATIONS

Device	xc6vlx240t-1ff1156
Kernel	AES-256 core
Latency	128 Clock Cycles
Channels	8
Clock	200 Mhz
Throughput	1.49 Gbits/Sec $\sim$ 11.02 Million Encryption/Sec
Slice Logic Utilization	
Number of Slice Registers	4448 out of 301440 1%
Number of Slice LUTs	5467 out of 150720 3%

V. SETUP

As a capability demonstration application for such system, 40-bit partial key brute force on AES-256 was planned. Thus we have implemented the proposed architecture using four ML-605 Virtex-6 FPGA Board. The FPGAs were arranged in a cluster formation as shown in a figure 10. For this setup we have reused AES-256 core, developed under our previous work [3] with the following specifications (see table:IV). In order to utilize maximum resources without compromising with timing issues we were able to accommodate 22 AES256 cryptographic kernel cores per FPGA with moderate device utilization of 65% (see figure:11).

VI. RESULTS

Comparison have been done between developed FPGA cluster with varying AES-256 crypto cores (kernels) and the AES-256 ECB implementations over x86 processors (see table:V).

Further, we have compared different architectures comprised of higher end GPUs, X86 Processors and this design. Since there are large variations between costs and power among these architectures, various normalised comparisons based on cost, power, cost and power(see figure:12) are done

Table V
AES-256 CRYPTO CORES VS AES-256 ECB ON X86 PROCESSORS

	Single AES core on one xc6vlx240t FPGA board	Multiple AES cores on one xc6vlx240t FPGA board	Cluster with 4 xc6vlx240t FPGA board	i3-3220 CPU @ 3.30GHz	i3-3220 CPU @ 3.30GHz AES-NI crypto extension
Boards	1	1	4	1	1
cores	1	22	88	2	2
Design Latency	128	128	128	X	X
Channels	8	8	8	X	X
Clock	200Mhz	200Mhz	200Mhz	3.30Ghz	3.30Ghz
Throughput Gbits/sec	1.49	31.04	124.2	0.71	1.80
Encryption Throughput Million block enc/sec	11.02	242.5	970.6	5.95	15.13

Table VI
AES 256 ENCRYPTION THROUGHPUT, COST, POWER FOR DIFFERENT ARCHITECTURES

s/n		Thput Gbps	Thput Gbps (Extrapolated for AES256 ECB 14 rounds)	cost $	Power watt(W)	Thput /cost Gbps/$	Thput /Power Gbps/watt	Thput /(cost * Power) Gbps/($ * watt)
1	corei3	0.71		377	120	0.00188	0.00591	0.000015
2	corei3 ani	1.80		377	120	0.00477	0.01500	0.000039
3	Geforce GTX285	6.25 [1]	4.46	400	316	0.01116	0.01412	0.000035
4	Tesla C2050	60 [2]	42.85	5159	238	0.00830	0.18007	0.000034
5	vertex 6 FPGA[3]	31.04		1995	45	0.01556	0.69020	0.000345

[1] AES128 ECB implementation with 10 rounds [17]
[2] AES128 ECB implementation with 10 rounds [18]
[3] Xilinx vertex 6 (xc6vlx240t-1ff1156) based AES256 implementation with 22 cores (this design)

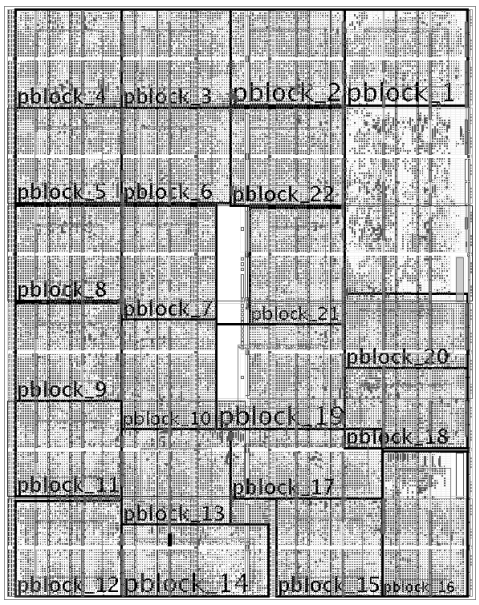

Figure 11. FPGA (xc6vlx240t) Floor plan for implemented 22 core AES256

(see table:VI). GPU results have been referred from the work of Q. Li et al. [18] and Nishikawa et al. [17]. Since these results were for AES-128 bit (10 rounds) implementation thus the results have been extrapolated for AES-256 bit (14 rounds) for comparision.

With this cluster AES-256 40-bit partial key attack was completed just under 10 minutes.

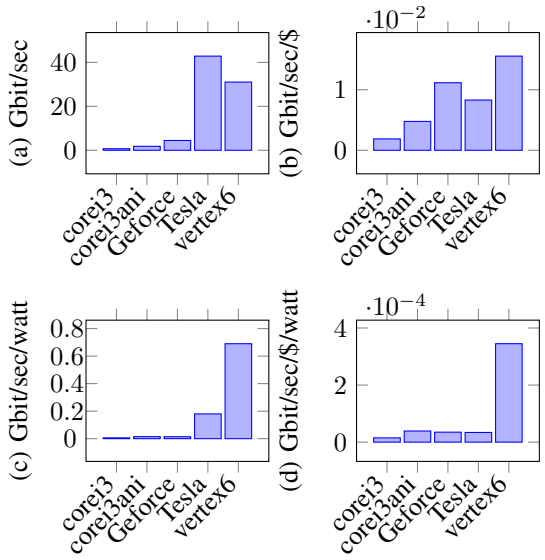

Figure 12. (a) Throughput, (b) Throughput/cost, (c) Throughput/power, (d) Throughput/(cost x Power)

VII. Conclusion

We have reported a development framework library FH-PClib for developing high-performance cryptanalysis applications based on FPGA cluster. This library accelerates the design process by abstracting underlying low-level details and provides high-level HDL programming interface. The library can be further developed and generalised in such a way that an end-user need not be required to know the finer details of the cluster, framework or even underlying FPGA technology.

Further we have also reported a scalable parallel FPGA cluster framework for cryptanalysis applications. This framework is suitable for many computation-intensive applications of different sizes and complexity. The flexibility provided enables it to be used with various FPGAs supporting different kind of high-speed interfaces. A user can configure various cluster parameters and node counts according to the computational requirements.

References

[1] G. Paul, S. Rathi, and S. Maitra, "On non-negligible bias of the first output byte of rc4 towards the first three bytes of the secret key," *Designs, Codes and Cryptography*, vol. 49, no. 1-3, pp. 123–134, 2008.

[2] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. Schuldt, "On the security of rc4 in tls." in *Usenix security*, vol. 2013. Washington DC, USA, 2013.

[3] A. Bajpai, B. Bathe, S. Parulkar, and A. Apte, "Design and development of an optimised hardware encryption module of gigabit throughput base on composite glois field arithmetic with feedback modes," 2008.

[4] C. Chang, J. Wawrzynek, and R. W. Brodersen, "Bee2: a high-end reconfigurable computing system," in *in IEEE Design & Test of Computers*, 2005, pp. 114–125.

[5] R. Baxter, "et al., maxwell - a 64 fpga supercomputer," in *in Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems*, 2007, pp. 287–294.

[6] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, "Breaking ciphers with copacobana – a cost-optimized parallel code breaker," in *Workshop on Cryptographic hardware and embedded systems CHES 2006, Yokohama*. Springer Verlag, 2006, pp. 101–118.

[7] T. Gneysu, T. Kasper, M. Novotny, and C. Paar, "Cryptanalysis with copacobana," *IEEE Transactions on computers*, vol. 57, no. 11, pp. 1498–1513, 2008.

[8] O. Mencer, K. H. Tsoi, S. Craimer, T. Todman, W. Luk, M. Y. Wong, and P. H. W. Leong, "Cube: A 512-fpga cluster," 2009.

[9] K. H. Tsoi, W. Luk, and S. Implementationgeneral, "Axel: a heterogeneous cluster with fpgas and gpus," in *Journal of Parallel and Distributed Computing*, vol. 69, no. 6, 2009, pp. 532–545.

[10] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient cnn implementation on a deeply pipelined fpga cluster," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 326–331.

[11] G. Marcus, W. Gao, A. Kugel, and R. Männer, "The mprace framework: An open source stack for communication with custom fpga-based accelerators," in *Programmable Logic (SPL), 2011 VII Southern Conference on*. IEEE, 2011, pp. 155–160.

[12] A. Stevens, "Introduction to amba 4 ace," *ARM whitepaper, June*, 2011.

[13] A. AMBA, "4 axi4-stream protocol," 2010.

[14] J. Decaluwe, "Myhdl: a python-based hardware description language," *Linux journal*, vol. 2004, no. 127, p. 5, 2004.

[15] K. Opencl and A. Munshi, "The opencl specification version: 1.0 document revision: 48."

[16] Altera, "Altera sdk for opencl," 2016.

[17] N. Nishikawa, K. Iwai, and T. Kurokawa, "Granularity optimization method for aes encryption implementation on cuda," IEICE technical report. VLSI Design Technologies (VLD2009-69), Tech. Rep., 2010.

[18] Q. Li, C. Zhong, K. Zhao, X. Mei, and X. Chu, "Implementation and analysis of aes encryption on gpu," in *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, June 2012, pp. 843–848.