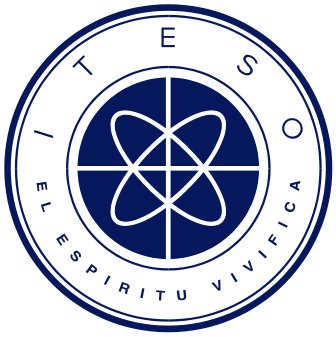
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



INTEGRACIÓN DE FPGAS Y MOTORES DE BASES DE DATOS

Trabajo final que para obtener el diploma de

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Zaira Lorena Zermeño Segura
Presenta: Oscar Enrique Valdés Moreno

Asesor: Raúl Campos Rodríguez

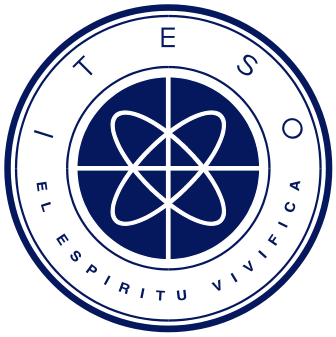
Tlaquepaque, Jalisco, Agosto de 2017.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



FPGAS AND DATABASE ENGINES INTEGRATION

Final report to earn the diploma of

EMBEDDED SYSTEMS SPECIALIST

Present: Zaira Lorena Zermeño Segura
Present: Oscar Enrique Valdés Moreno

Advisor: Raul Campos Rodríguez

Tlaquepaque, Jalisco. August of 2017.

Agradecimientos

Agradecemos a nuestras familias por el apoyo que nos han brindado a lo largo de la elaboración de este proyecto, en especial a nuestros padres que nos dieron la oportunidad de estudiar esta especialidad y expandir así nuestras oportunidades en el ámbito laboral y académico. Agradecemos al Dr. Raúl Campos Rodriguez y al Dr. Adrián Navarro Díaz que nos guiaron y asesoraron en el desarrollo de este trabajo, y además por todas las aportaciones que hicieron para el mejor desempeño de este proyecto. De igual manera queremos agradecer a nuestros compañeros de la especialidad en sistemas embebidos del ITESO, por sus consejos y ayuda; así como a todo el equipo de maestros que conforman el equipo de enseñanza de este programa.

Agradecemos al CONACYT por las becas No. 372319 y No. 373510 que nos otorgaron durante estos estudios.

Thanks to

We thank our families for the support they have given us throughout the development of this project, especially our parents who gave us the opportunity to study this specialty and be able to expand our opportunities in professional and academic environment. We thank Dr. Adrián Navarro Díaz and Dr. Raúl Campos who guided and advised us in the development of this assignment, and also for all the contributions that they made to the best performance of this project. Likewise, we would like to thank our colleagues in the ITESO embedded systems specialty, for their advice and help; and also to the all our teachers during this program.

We thank CONACYT for the scholarships No. 372319 and No. 373510 that we received during this studies.

RESUMEN

Este trabajo se enfoca en la integración de un FPGA a un motor de bases de datos. La finalidad es el diseño e implementación de algoritmos acelerados por hardware que sean utilices para las demandas de trabajo en tiempo real de las bases de datos modernas en el mundo de la ciencia de datos y el análisis del “BigData”. El trabajo propone una arquitectura de integración de un FPGA comercial y una base de datos de código abierto. Como ejemplo de aplicación se propone la encriptación y des-encriptación de datos mediante el uso de estos servicios implementados en un FPGA. El algoritmo utilizando es el públicamente conocido AES (Advanced Encryption Standard) de 128-bits. El algoritmo acelerado por hardware se utiliza en el motor de base de datos de código abierto MySQL mediante el uso de una función UDF en el API proporcionada por el motor a fin de probar algoritmos implementados en hardware para procesamiento de grandes cantidades de datos. El FPGA es una versión comercial del fabricante Xilinx.

ABSTRACT

This work focuses on the integration of an FPGA with a database engine. The purpose is to design and implement hardware-accelerated algorithms that are useful for the real-time work demands of modern databases in the world of data science and “Big Data” analysis. The work proposes an integration architecture of a commercial FPGA and an open source database. As an example of application, it is proposed the encryption and decryption of data using these services implemented in an FPGA. The algorithm used is the well-known 128-bit Advanced Encryption Standard (AES). The hardware-accelerated algorithm is used within the MySQL open-source database engine by using a UDF function in the API provided by the engine to test the algorithms implemented in hardware for processing large amounts of data. The FPGA is a commercial version of the manufacturer Xilinx.

TABLE OF CONTENTS

1. INTRODUCTION 9

1.1. CONTEXT 10
1.2. DATABASE MANAGEMENT SYSTEMS 11
1.3. OBJECTIVES 12
1.4. FUNCTIONAL DESCRIPTION 13

2. BACKGROUND 14

2.1. TECHNICAL FOUNDATION 15
2.2. THE MYSQL USER DEFINED FUNCTIONS (UDF) 20
2.3. FIELD PROGRAMMABLE GATE ARRAY (FPGA) 23

3. PROPOSED DESIGN 26

3.1. PROPOSED SOLUTION 27
3.2. BLOCK DIAGRAM 27
3.3. DESIGN DESCRIPTION 28
3.4. XILLYBUS IP CORE 29

4. IMPLEMENTATION 30

4.1. IP CORE FUNCTIONALITY 30
4.2. UBUNTU OS DRIVER FUNCTIONALITY 31
4.3. AES ENCRYPTION/DECRYPTION CUSTOM LOGIC 31

5. CONCLUSIONS 34

5.1. CONCLUSIONS AND DISCUSSIONS 35
REFERENCES 36

FIGURES LIST

Fig. 1. The MySQL most important blocks for high-performance DBMS. The Query cache, Parse and Optimizer are the key components in the core while the storage engine is of primary importance. MySQL provides a pluggable storage engine architectures. The MEMORY storage engine is a PDB retains the data in memory and is suitable for RTDBMS [12]..... 16
Fig. 2 MySQL Server connection flow. The instructions issued by a client traverses complex execution paths through the SQL engine [12]..... 18
Fig. 3 Dynamic Engine of the Execution in a DBMS [12]. The Employees Sample Database is used to teste the implemented functions [15]..... 19
Fig. 4 Flow diagram of the code function calls in the execution of an SQL statement that involves a generic UDF. The “xxx()” stands for the specific user-defined function name..... 22
Fig. 5 Xilinx FPGA Development Platform. (Creative Commons) – (CC)..... 24
Fig. 6. The KC750 FPGA base board conceptual block diagram. It populates a variety of elements that makes it a flexible platform for fast prototyping and proof of concept in different fields. It includes two blocks of flash memory, an eight lane PCI-E connector, 10/100/100 Ethernet interface, DIP switches and leds..... 25
Fig. 7 MySQL executions flow. The SELECT instruction is one of the most complex task performed by the SQL engine. 28
Fig. 8 Completed proposed architectural solution. From top to down, the design includes the user specific application, the MySQL engine, the Ubuntu OS, the FPGA development board connected through the PCIe port and the custom logic implemented in hardware..... 28
Fig. 9 XillyBus IP Core close-up..... 30
Fig. 10 AES Design blocks..... 32

ACRONYMS AND SYMBOLS

UDF	User Defined Function
FPGA	Field Programmable Gate Array
AES	Advanced Encryption Standard
DBMS	Data Base Management Systems
IMDB	In Memory Data Base
NVDIMM	Non-Volatile Dual In-line Memory Module
CPU	Central Processing Unit
SIMD	Single Instruction Multiple Data
API	Application Programming Interface
VHDL	VHSIC Hardware Description Language
FIFO	First In, First Out
PCI-e	Peripheral Component Interconnect Express
DDR3 SDRAM	Double Data Rate type three Synchronous Dynamic Random-Access
TCP/IP	Transmission Control Protocol/Internet Protocol
DMA	Direct Memory Access
SO-DIMM	Small Outline DIMM
UART	Universal asynchronous receiver/transmitter
USB	Universal Serial Bus
PDB	Pluggable Data Base

1. INTRODUCTION

This work focuses on the design and integration of FPGA technology to the Database Management Systems. As an application example, this work implements in the FPGA hardware encryption and decryption algorithms that the DBMS is able to use. An open source database engine was used to the integration of the proposed design. The data is processed using a commercial version of Xilinx FPGA.

1.1. Context

The database applications have notably increased its market presence in the recent years. Examples of real Data Base Management Systems (DBMS) are e-commerce, online reservation systems, banking, and mobile advertising networks among others. A DBMS provides data repository services by efficient storage mechanisms, and gives the user functionalities for data manipulation and presentation. On the other hand, the tasks in a real-time system include time restrictions. Thus, usually, a DBMS has time constraints for attending data base transactions. Typically, the real-time constraints deal with the deadline times and time to be ready for the system tasks, which are usually hard to ensure. There real-time systems classify in hard and soft. By the one hand, in hard real-time systems, the time constraints are a must, such as in many automotive security subsystems. In this kind of systems, the fail to meet the real-time constraints is catastrophic. On the other hand, in soft real-time systems, the time constraints are goals, such as in cell-phone communication network. Usually, a fail to meet the real-time constraints is not catastrophic in this case. Most of the commercial DBMS classifies as soft real-time systems [1]. The DBMS software vendors have implemented a variety of techniques for reducing the latency of database transaction operations and increase its performance. The technique includes highly optimized disk accesses, memory alignment of data, query optimization and others. In Memory Database (IMDB) Systems take advantage of modern highly efficient and cheap main memory systems and relies the entire data storage to main memory. It is contrasted with database management systems that employ a disk storage mechanism. Obviously, IMDB are various orders of magnitude faster than disk-optimized DB. This is natural since main memory accesses are thousands of times faster than disk accesses. Moreover, the internal optimization algorithms are simpler and execute fewer CPU instructions in IMDB than in traditional DBMS. Moreover, accessing data in main memory eliminates disks seek time when querying the data, which provides faster and more predictable response time [3]. Thus, IMDB brought upon important changes in data representation, algorithms for data access, query processing, recovery, and concurrency control [4]. The major attracting benefits of an IMDB are accelerated transactions, high reliability, data integrity, multi-user concurrency with quite consistent response times independently of the operating system it runs on. Additionally, with the introduction of non-volatile NVDIMM technology, IMDB can run at high speed and keept the data if an event of power failure occurs, as well. Also, IMDB are often used as a “cache” system for traditional DB. For example, Oracle provides TimesTen, an IMDB which supports relational databases, with persistence and recoverability. All data within the database is located in physical memory (RAM), which means that no disk I/O operation is required for any data operation [5].

This work deals with the application of hardware acceleration techniques on FPGA to the application of Database Management Systems (DBMS).

1.2. Database Management Systems

The CPU vendors are including instructions working on several data elements in parallel in their modern designs. These instructions are mostly within the category of Single Instruction Multiple Data (SIMD) instructions, since they are capable of apply a single instruction to more than one data elements [6]. The SIMD technology of modern processors was initially intended to accelerate the performance of multimedia and related applications. The SIMD paradigm reduces compute-intensive loops by feeding more than one data per instruction cycle. Despite that the SIMD instructions were originally designed to accelerate the performance of applications with repetitive operations on large arrays of data, such as motion video, real-time physics and graphics, this technology also provides new opportunities for accelerating critical operation in other areas such as in database engines. There are some works dealing with the application of vector processing technique to the database field. For example, in [7] the authors investigate the specialized hardware methods for joint and sort operation acceleration in the RINDA relational database. There are other works that exploited the advantage of parallel processing as CPU level such as SIMD instructions or GPU's [8],[9]. Also, a work related with this project was implemented several algorithms based on the SIMD instruction set of a modern microprocessor architecture [10]. However, there are few works reported in the literature dealing with the integration of a FPGA and a database engine, which is the focus of this work. The motivation of this works is based on the idea that the FPGA technology can provide flexibility and hardware-based acceleration of demanding algorithms for the processing of the big amount of data. For example, with a combination of an FPGA within the DBMS core, the companies may implement custom logic for its most daily demanding operations. The DBMS vendors are free to provide a generic engine that could be customized in-situ depending on the company profile and requirements. Based on these goals, this work explores the integration of a Xilinx FPGA to the MySQL open-source DBMS.

One of the most used database engine is the MySQL project. One of the most important aspects of MySQL is that the project is open-source. It includes a vast variety of database related tools and engines for almost every DBMS activity. For example, it includes several storage engines such as InnoDB, MyISAM, and Memory, among others. It provides connectors to other engines such as JDBC, ODBC, .Net, etc. The list accessory tools include

workbench, replication, partitioning, backup, monitoring services, and scalability options for the enterprise, security, and audit, among other services.

The Database Server, an essential module of a database engine, provided in the MySQL project is scalable, easy to configure and fast in the execution of queries. This server can run on a simple desktop, or on an entire machine completely dedicated to the MySQL engine. The configurability of the server allows to take advantage of the system memory and CPU power, as well as the input/output capacity available on the target machine. Additionally, the database engine can also be scaled up to a cluster of machines.

The connectivity options, high speed, and security systems make MySQL Server extremely suited for accessing databases on the Internet and also for providing web services over it. The MySQL Database Software is based on a client and server paradigm that consists of a multi-threaded SQL server supporting different back ends, several client programs as well as development tools. Also, there is an embedded multi-threaded library that links into user application to get a smaller, faster, easier-to-manage standalone product [11]. All those services make MySQL a good choice for research as well as for the enterprise applications.

1.3. Objectives

The general objective of this work is to investigate the application of hardware acceleration technique on FPGA to the speedup of the critical functions on DBMS.

The specific objectives of this work are:

- Select a platform suitable for the implementation of hardware accelerated function for DBMS
- Propose functions that are suitable for its acceleration by means of hardware acceleration techniques on FPGA
- Design and implementation on FPGA technology of functions that could be useful for the functionality of a DBMS
- Design and execute performance test of the implemented functions

1.4. Functional Description

This project is devoted to investigate the application of hardware acceleration technique on FPGA for DBMS. The project should implement a prototyping platform suitable for the application of functionalities such as encryption, decryption, compression, decompression, table scans and so on, which are critical function in real DBMS. The prototyping platform must run Linux based operating system where the version of MySQL 7.2 DBMS can execute. The aim that this project try to investigate is the suitability of the FPGA accelerated functionalities against the native implemented versions. In this stage of the project, the FPGA implemented services should produce same results as native functions for a proof-of-concept prototype.

2. BACKGROUND

2.1. Technical Foundation

A query is an SQL statement having a meaning inside the engine. The Fig. 1 shows the execution of a connection in server module of MySQL. Typically, a Command Dispatcher forwards queries to the Parser through the Query Cache Module. The Query Cache Module checks if the query is of the type that must be cached and if there exists a valid previously computed “cached” result. In the case of a cache hit, the execution is short-circuited at this point. The cached result is returned to the user, and the Connection Thread receives control. Then, it is now ready to process another command. The behavior of the Query Cache Module is quite like the cache system in modern hardware processors. That is, if the Query Cache Module reports a miss, the query goes to the Parser, which will decide on how to transfer control based on the query type. For more information on the execution engine of MySQL see the reference manual [13].

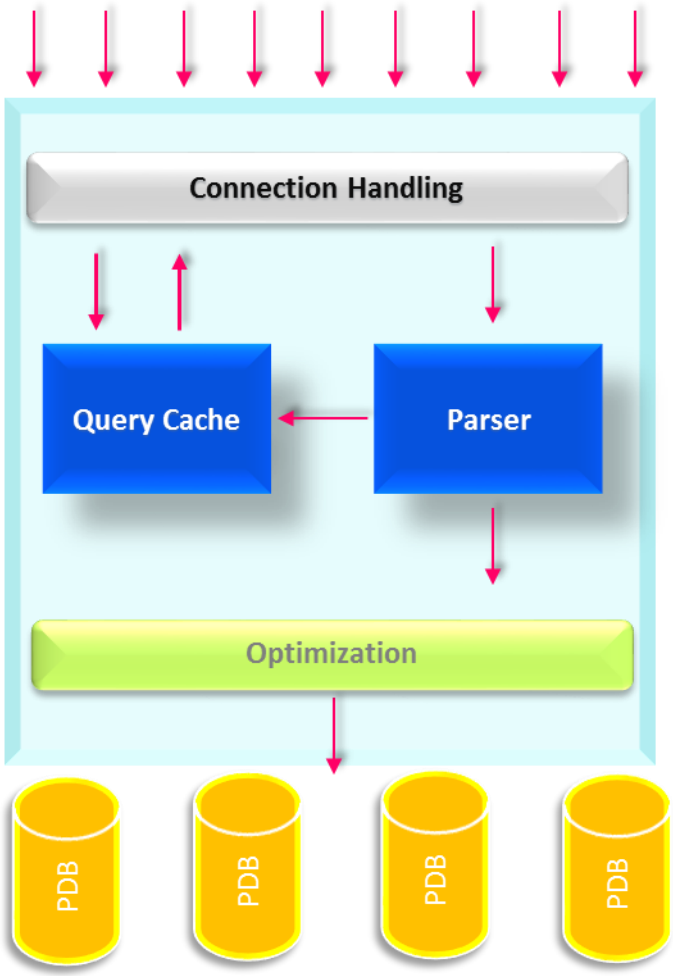


Fig. 1. The MySQL most important blocks for high-performance DBMS. The Query cache, Parse and Optimizer are the key components in the core while the storage engine is of primary importance. MySQL provides a pluggable storage engine architectures. The MEMORY storage engine is a PDB retains the data in memory and is suitable for RTDBMS [12].

For example, the MEMORY storage engine allows the creation of special-purpose tables which contents stores completely in main memory. This storage engine is suitable for data caching, or for operations involving fast access and low latency queries. The database design flow should consider a suitable approach that allows that the data can fit in memory. Typical situation includes a read-only or read-mostly data access pattern, with a limited number of update operations. The table in the MEMORY storage engine are allocated in small blocks. Tables always use dynamic hashing for inserts. The engine puts the deleted rows in a linked list that are reused when new insert operations into the table apply. See [13] for more information on the MEMORY PDB engine.

The tables in this storage engine use a fixed-length row-storage format. The variable-length types such as VARCHAR stores by using a fixed length field. The storage engine limits the maximum size of a memory table by the system variable max_heap_table_size. The default value of this variable is 16MB. A biggest table size is allowed changing the value of such a system variable. The tables in the MEMORY storage engine use hash indexes by default. This make the tables very fast for single-value lookups, and quite convenient when creating temporary tables. The storage engine supports BTREES indexes, as well. The indexes per table in the storage engine are limited to 64, while the maximum columns per index are 16 and a maximum key length of 3072 bytes. See [14] for more information about the tuning parameter of the MEMORY PDB engine. This work uses the MySQL 7.2 with the MEMORY storage engine as testing platform for a RTDBMS. It implements hardware accelerated functions that interact with the MySQL engine.

The SELECT queries are forwarded to the Optimizer and are by far the costliest queries in a DBMS. In MySQL, the Optimizer is responsible for creating the best strategy to answer the query, and execute such a strategy and deliver the result to the client or memory table. As usual, it is the most complex module in the MySQL code. The entry point to the optimizer is the function *mysql_select( )* which is implemented inside the file *sql/sql_select.cc*, at the core of the DBMS. The optimization process executes over a complex set of function calls, function pointer, pointer to pointer, and pointer assignment widely used. Even though, such a complex code design makes the maintenance of MySQL engine not a simple task, this also makes the core engine quite efficient. The optimization process is mainly spread over three files, such as mentioned, *sql/sql_select.cc, sql/sql_executor.cc,* and *sql/sql_optimizer.cc.* For further details about the optimization process in the execution of queries in the MySQL engine, see [14].

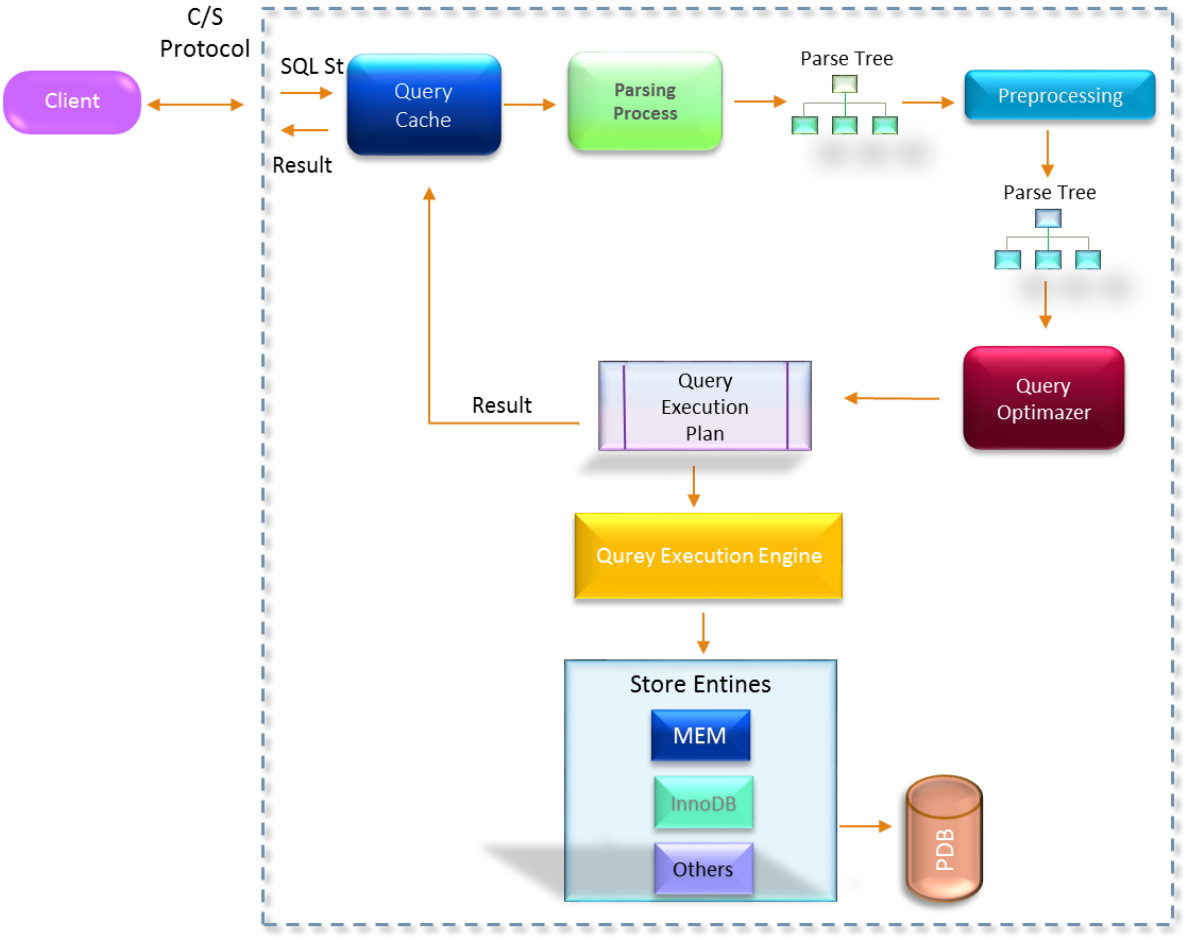


Fig. 2 MySQL Server connection flow. The instructions issued by a client traverses complex execution paths through the SQL engine [12].

The optimization process uses the major of the data structures in MySQL, such as *THD*, *JOIN_TAB*, *TABLE*, and *JOIN*, to mention a few. The optimization flow is executed through a set of stages such as *prepare( )*, *optimize( )*, *exec( )*, *make_join_statistics( )*, *find_best_combination( )*, and *optimize_cond( )*.

The *JOIN::exec( )* is the entry point for a SELECT statement execution. It includes the following main stages: *do_select(JOIN *join)*, followed by a call to the function *join->first_select(join, join_tab, 0)*. Then, it follows a call to function *sub_select(JOIN *join, JOIN_TAB *join_tab,**bool** end_of_records)*. Finally, a while-loop performs the required operation for all the records in the query. The Fig. 3 shows a diagram of the dynamics environment where the MySQL engine executes the queries of the DBMS.

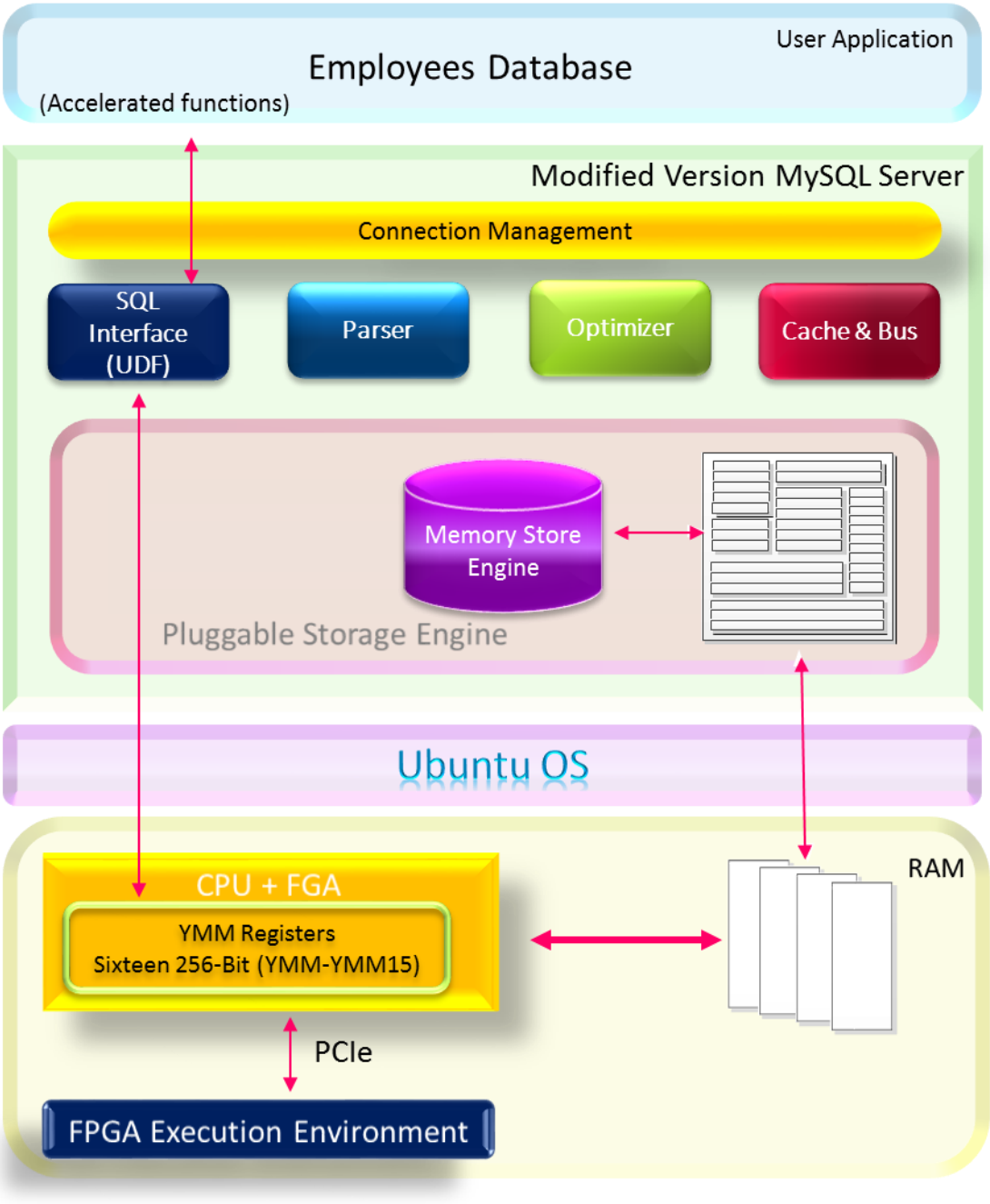


Fig. 3 Dynamic Engine of the Execution in a DBMS [12]. The Employees Sample Database is used to teste the implemented functions [15].

The execution of the function *ib_sum_XXX( )* is at the end of a chain of function calls. As mentioned, MySQL is quite efficient in part by the complex function call that otherwise may

require a lot of costly context switch which are prohibited for Real Time applications. However, it also reduces the impacts on the performance that user's functions can provide at such a nested level. That is, if it assigns 10% of the query execution time cost to each nested function call, then the user defined functions implemented in any technology, such as FPGA, can impact on the 10% of such an execution time cost. The Fig. 3 provides a conceptual diagram for the execution flow of a SQL statement once it has been syntactically analyzed and optimized.

2.2. The MySQL User Defined Functions (UDF)

The MySQL engine provides the user-defined function (UDF) interface. The UDF are compiled as object code and then inserted to and removed from the MySQL server dynamically using a built-in mechanism. The CREATE FUNCTION and DROP FUNCTION statements allows for the creation and destruction of user defined functions, respectively. Thus, the MySQL engine increases its functionality by the UDF interface which functions executes as the native built-in core functions. Rather that native functions which are compiled into the MySQL server and become available on a permanent basis, the UDF's modules can be plugged and unplugged on the fly. The specific mechanism depends on the target OS. For example, in the Linux-based distributions, the UDF's are compiled as shared object libraries. The *"plugin"* directory in the MySQL installation path holds the shared object libraries where the UDF's are implemented. Once a UDF is plugged to the MySQL engine, it can be invoked in every SQL statement just like any native function. A UDF can return string, integer, or real values and can accept arguments of those same types. Any UDF implemented within the MySQL engine can operate on a single row at a time, or on groups of rows in the form of aggregate functions. The core of a UDF can check the number, types, and names of the arguments passed to them. As well, it can tell MySQL core to coerce arguments to a given type before passing them to the function. Additionally, a UDF can indicate to the core that a function returns NULL or that an error during its execution has occurred. In the version of the UDF interface used in this work, the functions must be written in C or C++, and the target operating system where MySQL is supposed to run must support dynamic function loading. The MySQL source distributions include the header file *include/mysql_com.h* where symbols and data structures related to the implementation of UDF's are defined.

To simplify the implementation process of a UDF, there is a natural equivalence between SQL data types and C/C++ data types. That is, the SQL *string* is equivalent to *char pointer*

in C/C++. Similarly, *integer* and *real* data types in SQL are equivalent to *long long* and *double* data types in C/C++, respectively. The MySQL engine handles the UDF accordingly to the flow diagram of the Fig. X.

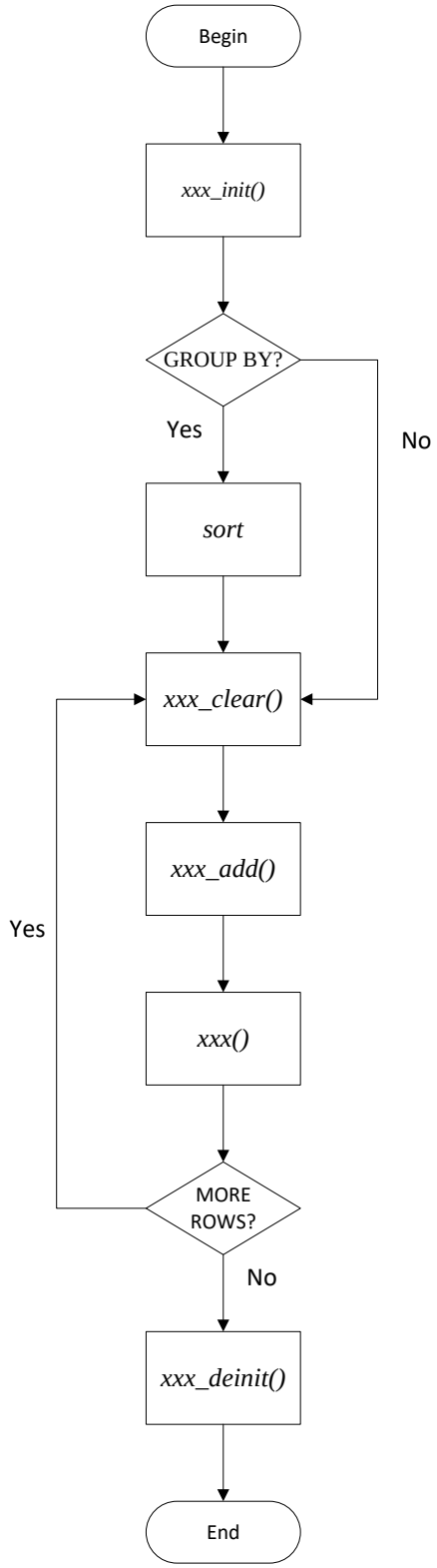


Fig. 4 Flow diagram of the code function calls in the execution of an SQL statement that involves a generic UDF. The "xxx()" stands for the specific user-defined function name.

Briefly, it calls to the *xxx_init()* method to let the UDF to allocate memory. Then the engine sorts the table according to the GROUP BY expression, if any. After that, the *xxx_clear()* method is called for the first row in each new group, if any. Then, the core calls the *xxx_add()* method for each row that belongs in the same group. So, the engine calls the *xxx()* method to get the result when required. The MySQL core repeats from the third to the fifth steps until all rows has been processed. Finally, the *xxx_deinit()* method is called to let the UDF free any memory it has allocated.

Thus, the UDF must implement all the previous methods and they must be thread-safe. The *"xxx()"* stands for the UDF's name. This work uses the UDF interface for implementing function implemented with FPGA technology. For example, the experiments included the AES-128 encryption and decryption functions that are implemented in a Xilinx FPGA and used in the MySQL engine through the UDF interface.

2.3. Field Programmable Gate Array (FPGA)

The Field-Programmable Gate Array (FPGA) technology allows to the final users or OEM's for customizing electronic block configurations after the manufacturing process. The FPGA electronic custom blocks are generally designed by using a Hardware Description Language (HDL), such as VHDL, Verilog, or System-C. The versatility of the FPGA technology makes it suitable for application where reconfiguration or the ability of custom block configuration is a must. The KC705 evaluation board for the Kintex™-7 FPGA provides a hardware environment for rapid prototyping, developing and evaluating designs based on the Kintex-7 XC7K325T-2FFG900C FPGA. The board provides features common to many embedded processing systems, such as:

- DDR3 SODIMM memory
- 8-lane PCI Express® interface
- Rri-mode Ethernet PHY
- General purpose I/O
- UART interface.

The Kintex®-7 FPGA KC705 development board provides a high-performance development and fast-prototyping platform using the Kintex-7 FPGA family of processors. The specific

processor used in this work is a Kintex-7 XC7K325T-2FFG900C FPGA. The experimentation board populates 1 GB DDR3 memory SODIMM, 128 MB Linear BPI Flash memory and 128 Mb Quad-SPI Flash memory.

The development board is connected to a host PC through a PCI Express with eight lanes. This simplifies the interfacing process with the MySQL core engine. The Fig. 5 provides a view of the development board.

Fig. 5 Xilinx FPGA Development Platform. (Creative Commons) – (CC)

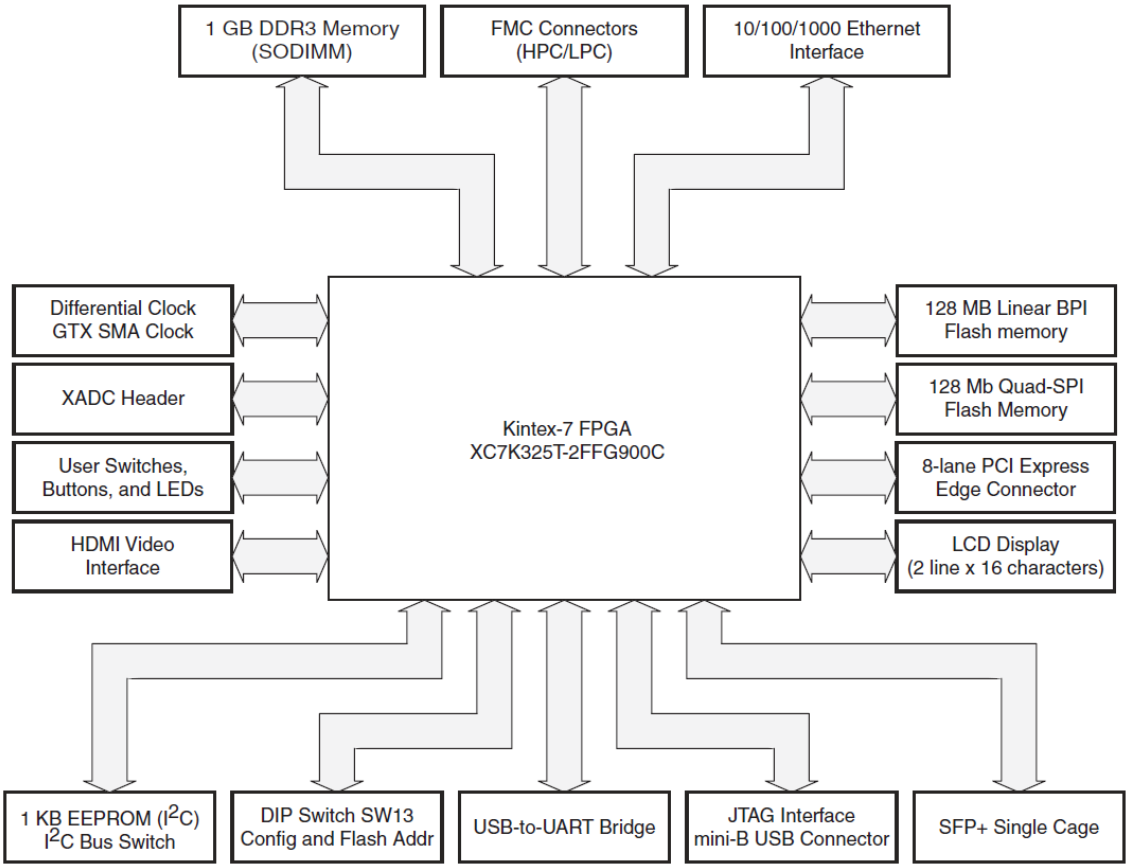


Fig. 6. The KC750 FPGA base board conceptual block diagram. It populates a variety of elements that makes it a flexible platform for fast prototyping and proof of concept in different fields. It includes two blocks of flash memory, an eight lane PCI-E connector, 10/100/100 Ethernet interface, DIP switches and leds.

The Fig. 6 depicts a conceptual block diagram of the KC750 board. The core is a Kintex-7 FPGA. The experimental board provides logic capacity, and signal processing modules for demanding high-performance applications, such as those presented in the processing of Big volume of Data.

3. PROPOSED DESIGN

3.1. Proposed Solution

The solution proposed in this work includes the integration of a FPGA development kit into Intel Core-i5 workstation with 2GB RAM DDR3 1333MHz, running Ubuntu OS and MySQL Cluster 7.2 suite.

3.2. Block Diagram

The Fig. 7 depicts a block diagram of the proposed solution. The Kintex®-7 FPGA KC705 card inserts into a free 8x PCI-E slot in the Core-i5 workstation. The Ubuntu 12.04 device driver configures the PCI-E communication to the Kintex®-7 FPGA KC705 card. Thus, the user space applications can communication with the FPGA through the PCI-E interface. Over this infrastructure there is the MySQL engine and the UDF interface.

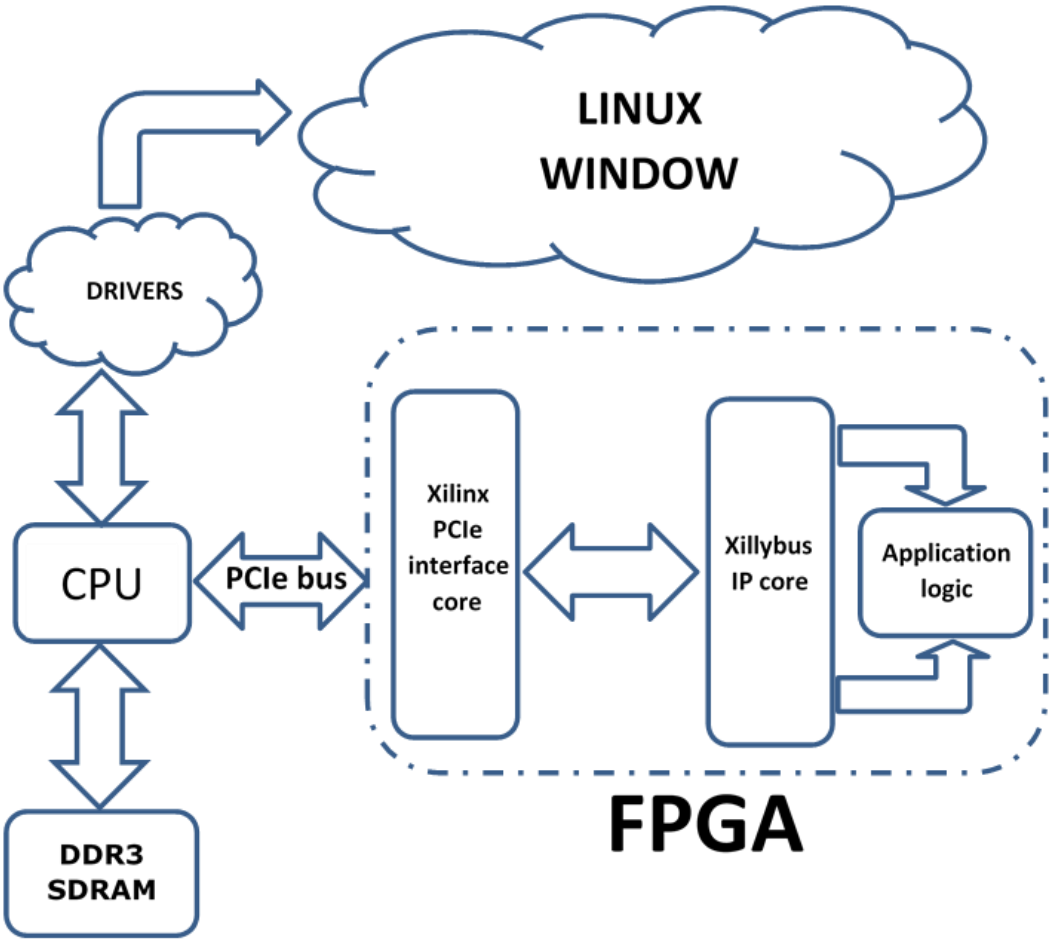


Fig. 7 MySQL executions flow. The SELECT instruction is one of the most complex task performed by the SQL engine.

3.3. Design Description

The main design consists in using a Test APP DB to encrypt and decrypt the contained data using the FPGA to execute all the required transactions related to AES algorithm. This is done through the UDF which invoke the required functions to send the data traffic to the XillyBus IP core using the PCI-e protocol, and in that way, encrypt/decrypt transactions are performed in parallel while the CPU is doing different tasks.

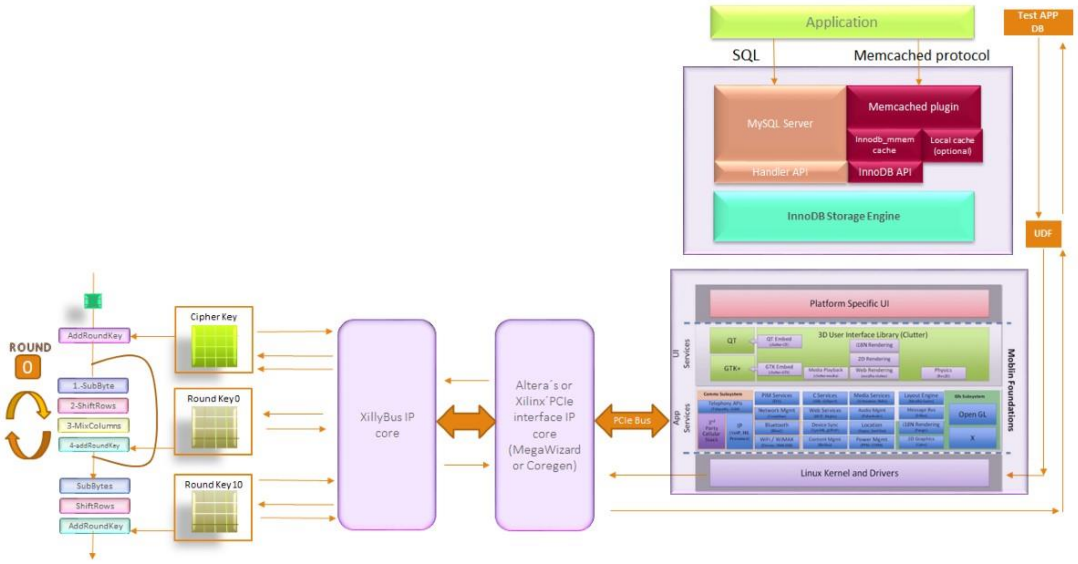


Fig. 8 Completed proposed architectural solution. From top to down, the design includes the user specific application, the MySQL engine, the Ubuntu OS, the FPGA development board connected through the PCIe port and the custom logic implemented in hardware.

3.4. XillyBus IP Core

Xillybus is not just a simple sheath for communication. Rather, it is a spectrum for several point-to-point stream pipes for data transactions. This is a completed solution that has heavy stress on numerous FPGA systems and IPs core configurations. It is necessary to emphasize that the Xillybus is very robust and the dependencies for XillysBus FPGA can be found on the datasheet. It is relatively a simple task develop a target application for testing and evaluation with low-risk assignment. Since the design of a specific PCIe protocol is out of the scope of this work, this work uses the open IP core provide by the community [17].

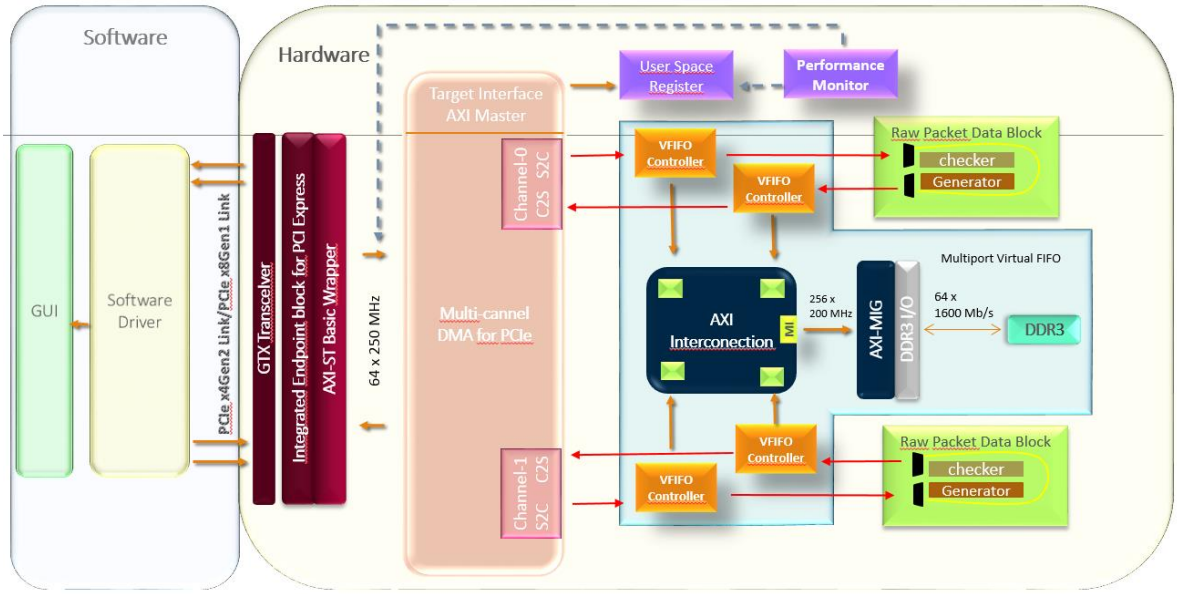


Fig. 5 Reference Design Diagram of the XillyBus IP Core [17].

4. IMPLEMENTATION

4.1. IP Core Functionality

XillyBus IP core sends data through a standard FIFO application which is supplied by the AES Encryption/Decryption UDF, and in this way the XillyBus IP manages the data traffic. Xillybus core verify the FIFOs phases that are designed to act like a stack with "empty" and “full” signals in a round-robin style.

The Xillybus IP core has no awareness about the expected data rate or when the user is suppliying it. In the other hand Xillybus IP core has no knowledge about the FIFO's current state. This does communication aught inefficient when the data rate is trending low, because little packages of data, small chunks, and short transactions are sent through the bus as soon as they reach the FIFO, instead than waiting for a specific chunk of data to get in. The relevance of this inefficiency is quite minimal because it does not apply at an increased data rate. With this design, Xillybus is quite efficient, and wastes bus transportation resources when they are not used for any data or task. The Fig. 9 shows a cluse-up of the Xullybus IP core. For more information about the custom IP core, see [18].

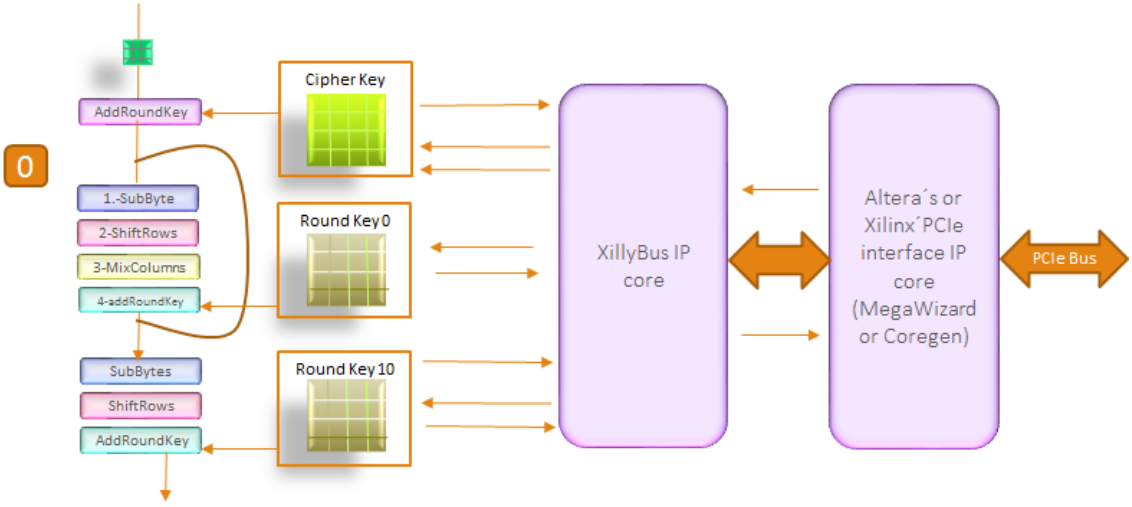


Fig. 9 XillyBus IP Core close-up.

4.2. Ubuntu OS Driver Functionality

The Ubuntu OS driver generates device files that conduct oneself like named pipes. The behavior of these files is like any regular file, but act very similarly to channels amid processes or TCP / IP streams. For the program running on the host, the difference is just that they run between both the other sides of the sequence is not another process but a FIFO in the FPGA. The Xillybus stream, alike a TCP stream, is designed to work with high-speed data transactions, as well as with unique bytes that arrive or are sent occasionally.

A binary driver supports any configuration of the kernel in the OS host where the Xillybus IP is intended to run. Based on this driver, the channels and their attributes are automatically detected and configured, as long as it is loaded on the host operating system. The device files are hence created. For another configuration of the core, see [18].

4.3. AES Encryption/Decryption Custom Logic

The Advanced Encryption Standard is a very reliable algorithm due to its portability and high security level. The Algorithm implementation simplify the use of FPGAs capabilities in order to accelerate the process for the data that needs to be encrypted/decrypted.

The AES algorithm is capable of handling 128-bit blocks, by using to this purpose several key sizes, such as 128, 192, and 256-bit. The AES algorithm has a very good flexibility and suitability to be implemented in hardware or software; and a relative simplicity of implementation process, since the operation involved are supported in the most popular target architectures.

The AES design comprises three different blocks, the AES-128, the AES-192 and the AES-256. Each block of the encryption/decryption algorithm may be used with public keys of 128-bits, 192-bits and 256-bits, for a total of nine different combinations. The Fig. 10 provides a layout of the blocks in the AES design. This work uses the block AES-128 with a 128-bits public key.

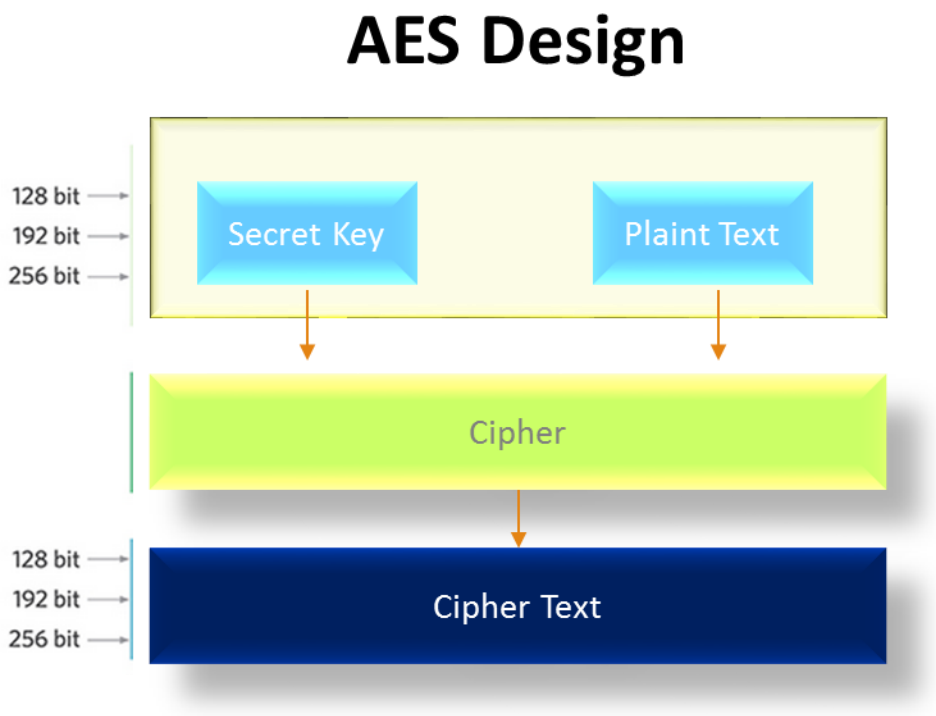


Fig. 10 AES Design blocks

Symmetric ciphers (also known as secret keys) use the same key to encrypt and decrypt, so the sender and receiver must know and use the same secret key. The implementation of the logic in the AES algorithm requires ten rounds for keys with 128-bits, twelve rounds for keys of 192-bit and fourteen rounds for keys with 256-bits. One round consists of several processing steps that include replacing, transposing, and mixing the incomming plaintext and transforming it to a final output text. These steps are commonly implemented by using matrix operations [19].

Briefly, the first step of the AES encryption process is to fill an array with the data to be encrypted. Then, the encryption operation transforms the array by repeating over several transforming rounds. The number of rounds is determined by the length of the key used in the encryption process. Thus, ten, twelve and fourteen rounds are required by 128-bit keys, 192-bit keys, and 256-bit keys, respectively.

The custom logic implemented in the FPGA used in this work was ten transformation rounds corresponding to blocks of 128-bits with public key of 128-bits as well. The column of Salary of the Employees Sample Database was encrypted by using the AES-128 algorithm

implemented in the Xilinx Kintex-7 FPGA and the results compared against the native AES function implemented in the core of the MySQL DBMS. The results were completely equivalent in both implementations.

5. CONCLUSIONS

5.1. Conclusions and Discussions

This work addressed the integration of hardware accelerated services for Database Management System (DBMS) applications. It used the FPGA technology for the implementation of custom logic blocks that provided services to a DBMS. It used Xilinx FPGA technology to implement in hardware functions that MySQL engine can use. For proof-of-concept purpose, this work implemented the AES encryption and decryption logic into the Xilinx FPGA and interfaced it with the Ubuntu OS through the PCI-E interface. This work implemented two UDF wrapper functions that called the in-hardware encryption services. This work shown that the integration of hardware-accelerated services for DBMS are suitable and provides cost-efficient solutions for time-critical operations. The future work considers the use of the QPI interface instead of the PCI-E for a complete solution where the CPU and FPGA could communicate at a maximums speed and with a cache-coherent mechanism.

The custom logic implemented in the FPGA used in this work was ten transformation rounds corresponding to blocks of 128-bits with public key of 128-bits as well. The column of Salary of the Employees Sample Database was encrypted by using the AES-128 algorithm implemented in the Xilinx Kintex-7 FPGA and the results compared against the native AES function implemented in the core of the MySQL DBMS. The results were completely equivalent in both implementations.

REFERENCES

[1]. Liu, F., Narayanan, A., & Bai, Q. (2000). Real-time systems.

[2]. Garcia-Molina, H., & Salem, K. (1992). Main memory database systems: An overview. IEEE Transactions on knowledge and data engineering, 4(6), 509-516.

[3]. Gupta, M. K., Verma, V., & Verma, M. S. (2014). In-memory database systems-a paradigm shift. arXiv preprint arXiv:1402.1258.

[4]. P. A. Deshmukh. (2011). Review on Main Memory Database. International Journal of Computer & Communication Technology, Vol 2(7), 54-58.

[5]. Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B. G., & Naughton, J. F. (2002, June). Middle-tier database caching for e-business. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data (pp. 600-611). ACM.

[6]. Rodríguez, P. (2002, May). A radix-2 FFT algorithm for modern single instruction multiple data (SIMD) architectures. In Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on (Vol. 3, pp. III-3220). IEEE.

[7]. Satoh, T., Takeda, H., Inoue, U., & Fukuoka, H. (1991, April). Acceleration of Join Operations by a Relational Database Processor, RINDA. In DASFAA (pp. 243-248).

[8]. J. Zhou and K. A. Ross, “Implementing Database Operations Using SIMD Instructions,” Proc. ACM Sigmod, Madison, Wisconsin, USA, 2002 June 4-6, pp. 1-12.

[9]. S. Meki and Y. Kambayashi, “Acceleration of relational database operations on vector processors,” Syst. Comp. Jpn., vol. 31, pp. 79–88.

[10]. Francisco David Covarrubias Padilla and Diego Iván Romero González. “Programming of Embedded Vector Processing Units for Real-Time Database Systems.” Final Project, Embedded System Specialization Program, 2014.

[11]. Pérez López, C., & López, C. P. (2004). MYSQL para Windows y Linux.

[12]. Pachev, A., & Pachev, S. (2007). Understanding MySQL Internals. O'Reilly Media, Inc.

[13]. Widenius, M., & Axmark, D. (2002). MySQL reference manual: documentation from the source. “O'Reilly Media, Inc.”.

[14]. Schwartz, B., Zaitsev, P., & Tkachenko, V. (2012). High performance MySQL: optimization, backups, and replication. “O'Reilly Media, Inc.”.

[15]. MySQL Employees Sample Database. [Online]. Available at: https://dev.mysql.com/doc/employee/en/

[16]. VC707 Evaluation Board for the Virtex-7 FPGA User’s Guide [online] https://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf

[17]. An FPGA IP core for easy DMA over PCIe with Windows and Linux. [online] http://xillybus.com/

[18]. XillyBus IP Core Brief Description. [online] http://xillybus.com/downloads/xillybus_product_brief.pdf

[19]. Schneier, B. (2007). Applied cryptography: protocols, algorithms, and source code in C. john wiley & sons.

[20].