International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 7, No. 1, March 2018, pp. 21~33
ISSN: 2089-4864, DOI: 10.11591/ijres.v7.i1.pp21-33

Embedded Hardware Circuit and Software Development of USB based Hardware Accelerator

Sanket Dessai, Sandeep G.
Departement of Computer Science and Engineering, M.S.Ramaiah University of Applied Sciences, Bengaluru, India

Article Info

Article history:

Received Dec 5, 2017
Revised Feb 2, 2018
Accepted Feb 15, 2018

Keywords:

Embedded Systems
FPGA
Hardware Accelerator
USB
Xilinx ARTIX 7

ABSTRACT

This paper focus on design and develop a Hardware Accelerator which can plug in to Universal Serial Bus of any modern low power low cost embedded development system to do complex processing in a plug and play development environment. Cryptographic algorithms, steganography and encoding decoding applications can use co-devices to accelerate performance. In this paper an implementation of a hardware infrastructure for computing though USB bus of any small scale embedded controller board. Execution engine of the accelerator will be an FPGA which is connected to a USB controller with DDR memory to store user data. FPGAs can perform the process faster than low power microcontrollers to solve such algorithms. For the implementation XILINX ARTIX 7 FPGA is used to off load the algorithm for faster processing. System also has a Cypress USB interface chip for offloading data path. Hardware also has a DRAM memory for dumping the data to be stored. Design also explores different futuristic features like interrupt connection for faster response path, shared memory architecture for hand shake mechanism and GPIO connection for implementation of faster interfaces for IO expansion.

Corresponding Author:

Sanket Dessai,
Departement of Computer Science and Engineering,
M.S.Ramaiah University of Applied Sciences,
#470-P, Peenya Industrial Area, Peenya Second Stage, Peenya Bengaluru (Bangalore)-560058.
Email: sanketdessai.cs.et@msruas.ac.in

1. INTRODUCTION

Hardware acceleration is a method of using additional hardware to perform some functions efficiently and faster than a conventional software running on a more general purpose processing. Examples of hardware acceleration include bit boundary bock transfer acceleration functionality in graphics processing units (GPUs) and regular expression hardware acceleration for spam control in the server industry. In conventional systems processors are sequential like instructions are executed one by one, and are designed to run general purpose algorithms controlled by instruction fetch. Hardware Accelerators improve the execution of a specific algorithm by allowing greater concurrency, having specific data-paths for its temporary computation of the system, and possibly reducing the overhead of instruction control. Modern processors are multi-core and often feature parallel SIMD units providing performance yields benefits [1], [2].

In this paper, design of a hardware accelerator for small form factor embedded designs are discussed in detail, its development of models are implemented and tested. Figure 1 shows high level components of a system with hardware accelerator. Normal software execution flow will start with program's assembly code/instructions start interacting with core CPU. Underlying software will control the execution path to the accelerator. Data path is implemented, as a shared memory using a memory controller taking care of both RAM and it can be an independent memory for accelerator and CPU and managing it separately.

Hardware acceleration is suitable for any repetitive, intensive key algorithm. Depending upon granularity, hardware acceleration can vary from a small functional unit, to a large functional block like MPEG algorithm FFT calculation etc [3].

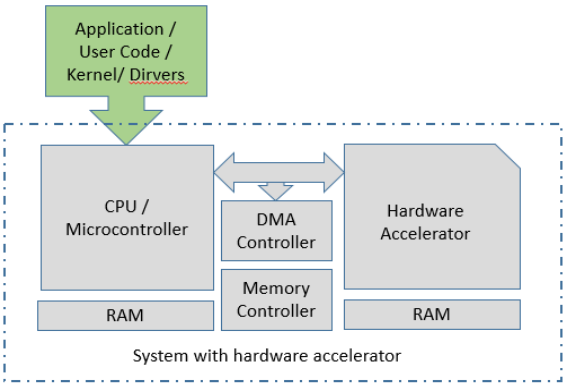

Figure 1. Computing with Hardware Accelerator

Fixed-function implemented on FPGAs, and fixed-function implemented on ASICs; there is a trade off between flexibility and efficiency, with efficiency increasing by orders of magnitude when any given application is implemented higher up that hierarchy. Small low power embedded designs are made to serve a particular functionality. For faster system performance embedded systems often use chip solutions for faster design time. In such cases the communication between the processor and its peripherals is not standardized that result in applications developed for one solution is not readily portable to another. Our implementation is to solve this problem, by offering embedded systems a programmable logic part in a generic USB bus which can move from design to design quickly. Currently there is no off the shelf product available for embedded systems to use as hardware accelerator. However, that such a system be implemented is fast becoming an imperative given the rise in embedded world's growth pace. There are many applications coming up in industrial and medical and internet of things segments which need processing power added for special algorithms [3], [4], [5].

1.1. Advantages of Hardware Accelerator

A customized processing unit outside the processor will be able to perform operation much faster than CPU of equivalent cost. Any added general purpose processor will out perform than a unit which can perform a specific task due to its design optimised model. Cost of processor is always a linear function of performance. Accelerators always give a better real time performance compared to generic processor, which does the task.

Accelerators are good in performing input output operation in real time applications. FPGA are even capable to implement high speed interfaces such as PCIE express connections. Dedicated hardware consumes lesser energy compared to the co processors or generic computing processors. External accelerators always improve performance for data streaming applications such as video audio encoding domains. It is keeps the cache memory dependency of CPU away from the performance. Bit level operations which are more frequently used in embedded application too will perform better on a dedicated hardware faster than a single controller. A single controller or processor can't do all the parallel task which are supposed to perform on a mission critical embedded system [1], [5].

1.2. Challenges and Advantages of Hardware Accelerator in Embedded Systems

The FPGA design acceptance path is relatively long and complex. The design cycle for FPGA based solutions are longer than software solutions due to the resource requirements. Considering the eco systems like OpenFPGA and OpenCL kind of software solutions along with FPGA IPs can solve this issue to a great extent. FPGAs are faster but they can only do only fixed task. Reprogramming capability added to the software eco system can solve this issue to a great extend even though it is time consuming. For an example to do beyond just format conversion like data stream filtering, FPGA will have to reprogram with newly added algorithm using new bit stream as shown in Figure 2.

FPGA design will be considered as a hardware design that by nature needed more planning and resources than software implementations. Testing time and process are more complicated in case of FPGA design software than traditional software. It requires time consuming synthesis and timing analysis to bring it to the final stage [6]. Hardware accelerated design need to be architected carefully, the designs which are not architecture well are traditionally will not do their use as shown in Figure 3.

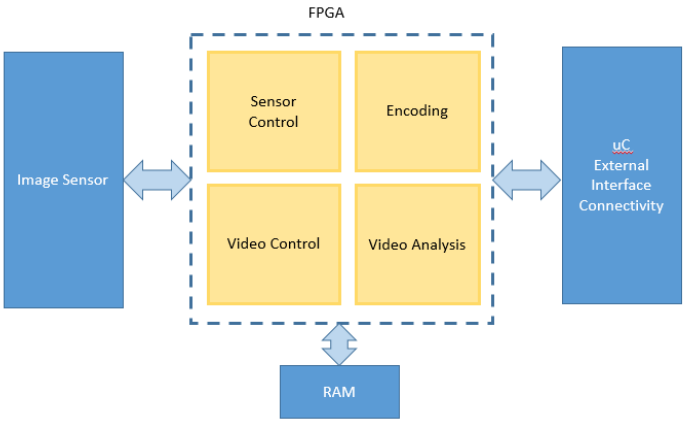

Figure 2. FPGA based IP Video Servillance

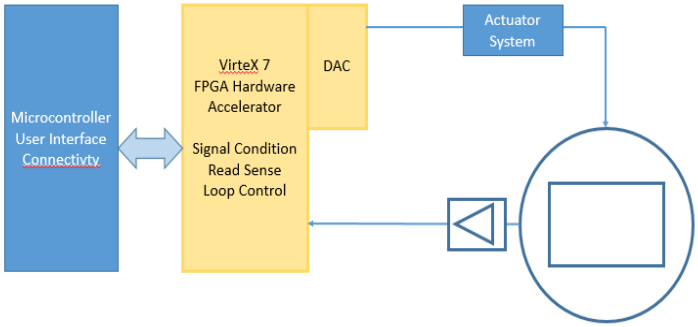

Figure 3. FPGA hardware accelerator IO based real-time control systems

Most of the embedded applications that are also real time applications which will demand very strict timing requirements. Therefore, an operating system that manages these tasks, have to behave in a deterministic and predictable way. The task scheduler also will have save predictability requirement which is major overhead. Many of the embedded devices operate on low power segments or in batteries, which will have many serial ports often used. These input output ports can create unnecessary overhead to the system. Major part of this overhead can be out sourced to the accelerator part in case of low power devices.

1.3. Challenges and Advantages of Hardware Accelerator in Embedded Systems

All the major embedded systems will have a general-purpose controller and associated hardware like memory, timers and associated components will be there on board or in ASIC. Depending on application requirements there can be other components also found like a system which is expected to perform signal processing tasks can have a multiply and accumulate (MAC), system which used cell phones have cellular network management related devices incorporated. Embedded system, which are designed for high-speed network access or memory data access, are likely to have DMA controllers and network interface controllers. Other than all these systems can have, special high-speed ports or codecs for also implement faster compression or decompression as shown in Figure 4.

Mathematical routines, which are implemented at a low level with an optimum customised parallel architecture, can realize using HW accelerators. This removes overhead of control, operating system, interrupts and interfaces. This implementation is highly deterministic and repeatable in delivering decisions or orders faster than software realizations in many cases [8], [9].

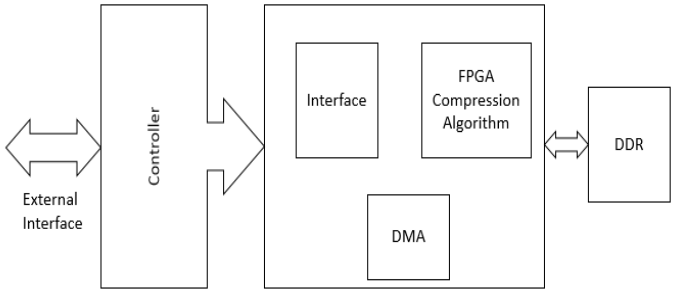

Figure 4. Compression algorithm in software-core

2. SYSTEM ARCHITECTURE DESIGN

In this section a design of system architecture is explained for the USB based hardware accelerator.

2.1. Circuit Design of FPGA Hardware Accelerator in USB

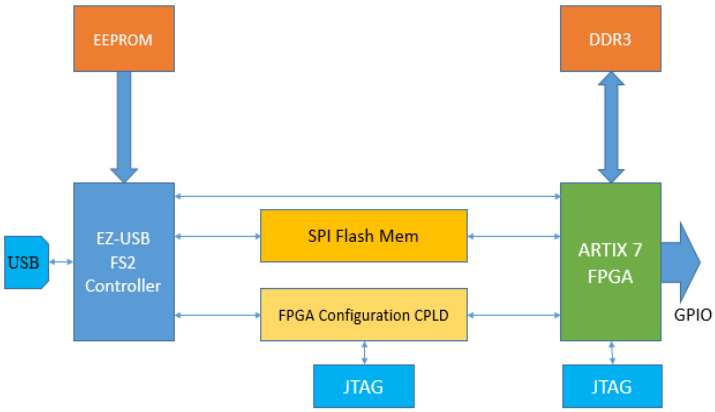

Figure 5. Block Diagram of hardware Accelerator Unit

The system will consist of interface to the computing modules and the FPGA execution engine. Device also will contain memory required for the data sharing and command exchange between host system and the accelerator. The design of the overall system contains EZ based USB interface and supporting EEPROM circuitry to configure the USB interface and for the microcontroller inside. CPLD allows a fast FPGA configuration through USB. It controls the configurations pins of flash and FPGA. Flash memory is given to store the FPGA bit stream file to help reprogramming if to use same algorithm to rerun.

Cypress EZ-USB FX2 and FX3 devices are low power, highly integrated USB microcontroller [10], [11]. They have a fully configurable General Programmable Interface and master/slave endpoint FIFO, which provides an easy connection to popular interfaces such as ATA, UTOPIA, EPP, PCMCIA, DSP, and most processors in a glue less way cypress also provides designer tools for users. FX3 devices provides a USB3 connection for faster data transfer as shown in Figure 5 and Figure 6.

The robust design of the circuit, provide easy to use plug and play capabilities kind of features of the Universal Serial Bus (USB) gaining great popularity for the interface system. USB has a wide range of bandwidth choices like low, high and Super speed modes makes it a choice of interface for a range of applications from slow speed peripherals like mouse keyboard sensors to high band width applications such as storage disks and scanners. Other than peripheral use, cases it is a proven interface for much high speed and slow speed industrial and medical applications [12].

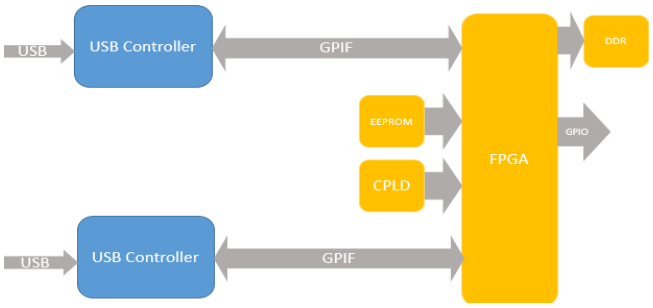

Figure 6. Block Diagram of Dual USB Hardware Accelerator

Table 1. Speed of Different Available Interfaces

Interface	Speed
FireWire 800	800 M bits/sec
USB 3.0	5 G bits/sec
USB 3.1	10 G bits/sec
eSATA	6 G bits/sec
Thunderbolt	10 G bits/sec
Thunderbolt 2	20 G bits/sec
Thunderbolt 3	40 G bits/sec
PCI Express 1x	4 G bits/sec (up+dn)
PCI Express 2x	8 G bits/sec (up+dn)

2.2. Design of CPLD and SPI Flash Design for Configuration

For customized applications, a microprocessor or CPLD can be used to configure an Aritx 7 either series device, Master SelectMAP mode or Slave SelectMAP mode. The 7 series FPGA Master SPI configuration mode enables the use of low pin-count, industry-standard SPI flash devices for bitstream storage. In this design, it is use a CPLD to select between EEPROM and FPGA device M [0-2] pins are selected using CPLD bitstream will be programmed to FPGA according to the selection. Data bus will be directed to CPLD DDPRM and FPGA according to the user selection as shown in Figure 7.

Through the firmware loaded through the EZ USB to its controller will be the master control of the application program interface. The internal micro controller will be used program the CPLD which takes care of the XLINX programming mode and EEPROM path. The firmware inside the micro controller plays a crucial role in it. Instead of complete implementation of logic inside one FPGA design will have to move the programming logic to the host machine which can break the portability of Hardware accelerator design [13], [14], [15].

A LGA package of 45 mm X 42.5 mm is designed with 1356 pins. On top of this package, the die (silicon) with the size of 10.34374 mm x 20.3797 mm is placed. Table 1 shows the details of the package and the die.

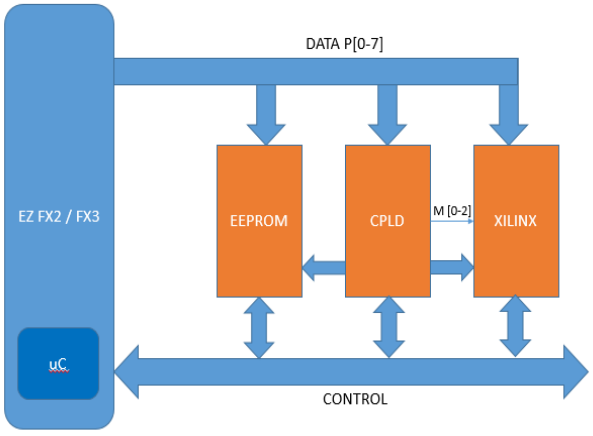

Figure 7. Block diagram of programming circuit

2.3. Design of Interconnect Logic between Blocks

There are multiple bocks in the system which needs data transfer in the system for programming ad for application data transfer SelectMAP signals are connected between CPLD and FPGA for faster programming support which is discussed in detail in the above sections. It has a general programmable interface connection between the FPGA and USB FX2 or FX3 device for faster data transfer between for applications as shown in Figure 8.

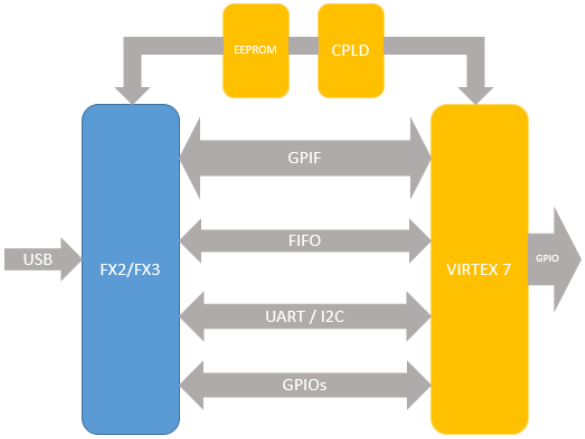

Figure 8. Interface routing diagram

2.4. Power Supply Design for the USB Hardware

Design of power supply and its management for FPGAs plays a crucial part of the overall product. In this hardware design low power high performance Xilinx Artix 7 FPGA for of loading the logic. Table 2, gives major voltage inputs and their spec for Artix 7 series.

Table 2. Voltage Levels of Artix 7

Xilinx Virtex-7 Power-supply Reqirements			
Power Rail	Nominal Voltage (V)	Tolerance	Description
V_{CCINT}	1.0*	±3%*	Voltage supply for the internal core logic
V_{CCAUX}	1.8	±5%	Voltage supply for auxiliary logic
V_{CCO}	1.2 to 3.3	1.11V to 3.45V**	Voltage supply for I/O banks
MGTAVCC	1.0	±3%	Voltage supply for GTX transceiver
MGTAVTT	1.2	±30mV	Voltage supply for GTX transceiver termination circuits

Other less important voltage levels are also there in FPGA such as VCCBRAM, VBATT and VREF that require lesser current. FPGA provide better programming control over CPLD and other devices. A FBG 96 DDR which is used for data sharing as major power consuming devices. Other than these, it need to consider EEPROM, FLASH and reset chip while considering the power requirements [15]. There are 3 major power requirements for the board 1 Volt 4 Ampere power supply for the FPGA internal logic and a 3.3 V 2 Ampere supply for the Input Output connection of the FPGA circuit. There is a separate 1.5V 2A power supply dedicated for DDR circuit. Following sections explains the details of each of these power supply sections in detail.

For 3.3V and 1.5V it is using AOZ1050 DC DC Buck Regulators, is a high efficiency, easy to use, 2A synchronous buck regulator as shown in Figure 10. The AOZ1050PI works from 4.5 V to 18 V input voltage range, and provides up to 2 A of continuous output current with an output voltage adjustable down to 0.8 V. If design, follow single power supply for the complete solution circuit will have to support a design of 20A power supply. So in this design a splitting of the power supply to multiple parts according to the requirement. There is also an option in circuit to switch between USB and external power supply to run the FPGA in faster mode. RT8288A is a synchronous step-down regulator with an internal power MOSFET as shown in Figure 11. It can source upto 4A of continuous output current over a wide input supply range with good load regulation and line regulation as shown in Figure 9.

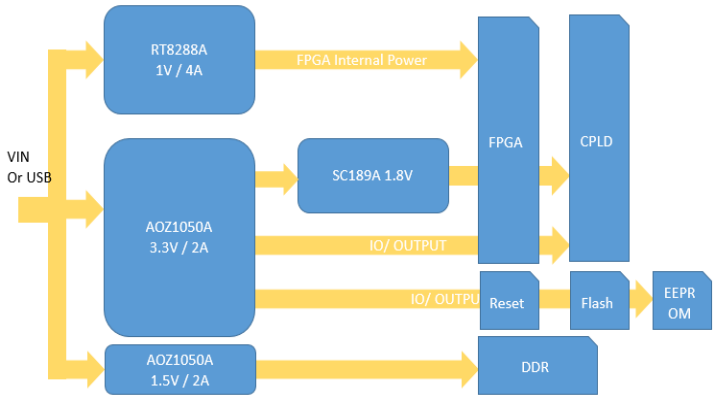

Figure 9. Diagram of power supply section for the accelerator

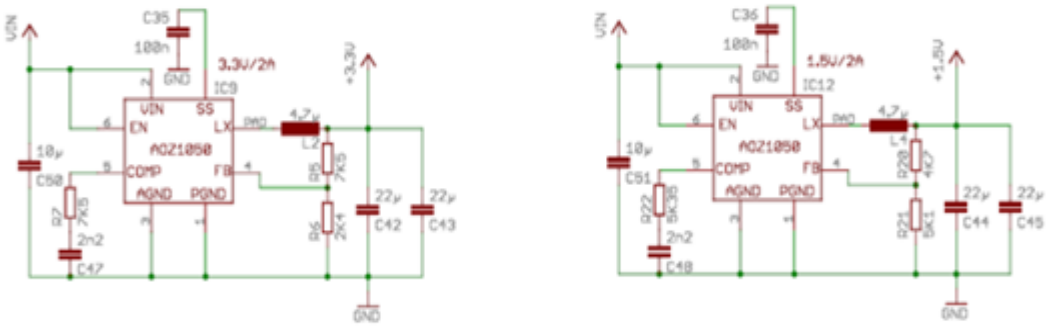

Figure 10. Schematic diagram for 1.5V and 3.3V using AOZ1050

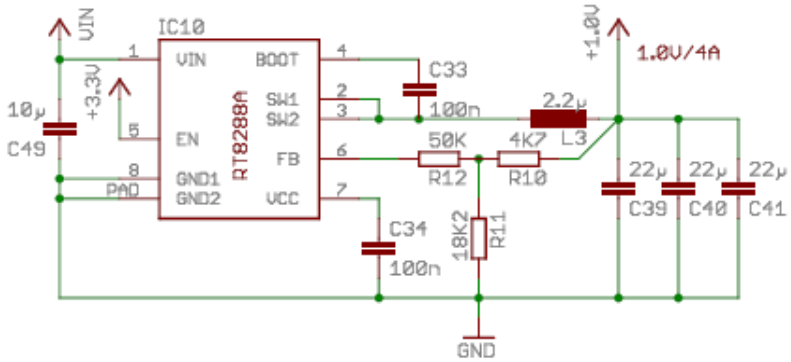

Figure 11. Schematic diagram for 1.0V supply using RT8288

3. DESIGN OF SOFTWARE COMPONENTS

Target or hardware accelerator in this deign need multiple software components to work together to achieve the performance throughput. Following picture illustrates different software components and there interoperation. GPIF interface can be designed using Cypress design tool. There will be a role for firmware to manage the CPLD also. Common interface code, which can be reused across designs. EZ FX firmware code will be fixed for all the designs. Programming code will flexible enough reconfigure in all the applications. FPGA interface code will be a common repository, which will have to integrate with applications as shown in Figure 12.

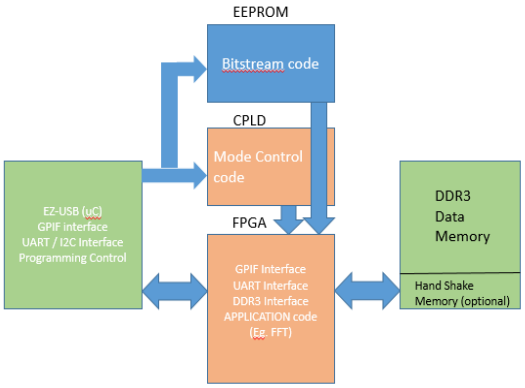

Figure 12. Software Components Interconnection in Target Hardware

Depends on the interface chip accelerator can use DMA Quantum FIFO mechanism used inside the controller chip for bulk data transfer. FX3 chip has a DMA engine inside which can be used for copying DDR data through FPGA to the SUB interface without the controller cycle time. In case of FX2 accelerator will have to make the system work with Quantum FIFO in auto configuration. Design has a dedicated memory region reserved for implementing the hand shake mechanism. User code result can be informed to the host through VHD code in the FPGA which is keep monitoring the shared memory region. This can be an optional model depends on the user code working model.

3.1. Open MP Solution for Embedded Host

OpenMP 4 support a wide range of compute devices Xeon Phis, GPUs, FPGA. OpenMP is use high level directive based solution by its architecture which helps the developers to spin the applications faster [16]. In OpenMP threads communicate by sharing variables for synchronizations to control race conditions. Even though none of the solutions are perfect industry adoption is critical. It supports a standard for shared memory parallel programming for industry. Programmers always need a write once, and reuse kind of architecture to avoid rewriting from basic and to provide incremental path essential for application codes to migrate. To extend the usage of a high-level programming model, OpenMP to multicore embedded systems and address the architectural challenges, software will have to use a runtime library, which understands the systems, and it should follow MCAPI as shown in Figure 13.

In recent developments embedded applications are becoming getting complex day by day and looking for code re usage as like high level desktop applications due to fats growth in hardware platforms technologies and applications. Here we are trying to analyse a model with of software APIs and hardware [17]. Even for highly flexible and efficient hardware design also need support of programming models and tools.

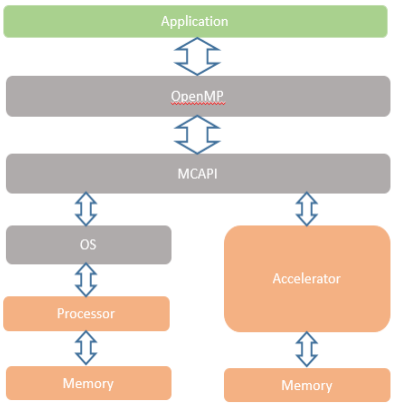

Figure 13. OpenMP MCAPI software block diagram for FPGA hardware accelerator

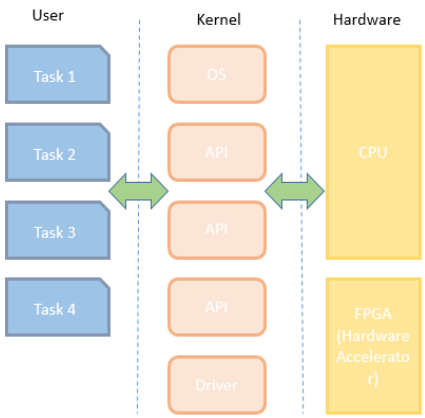

Figure 14. Abstraction in interface level for design

Physical hardware design can consist any kind of processors or hardware accelerator IP which will get wrapped using a software API level for overall functional completion. An extra support from drivers and operating system to get the final layers operational. The final target is to establish an interface layer between hardware accelerators, processors and application layer and maximum offer performance and memory management as shown in Figure 14.

OpenMP is a collection of APIs for multi-processor programming languages in C++ [18]. A normal computer running Linux or Windows can be used for application development and FPGA synthesis. And can use the bitstream along with driver applications to integrate with APIs to show the performance advantage. Use applications developed in a desktop environment allows faster debugging and integration. Application code can be transferred to FPGA or distributed between systems.

3.2. Host Application in Basic C Code

All the Embedded development environment will not be having luxury to use little heavy environment like OpenMP or other APIs. To address this situation, we are exploring one more option here to implement the hardware accelerator for small scale embedded systems which has a USB inter face to connect the designed hardware [19]. Figure 16 shows flow chart of application software for C implementation.

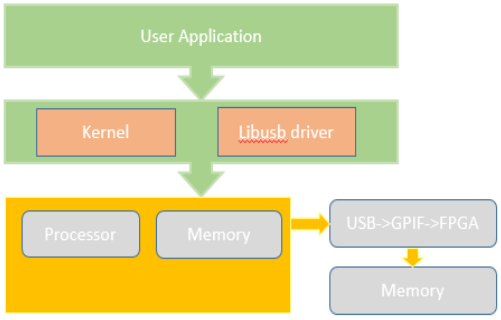

Figure 15. Software Implementation

Figure 16. Basic Software flow diagram

This design is using libusb, a library that provides applications access for data transfer through USB interface, which is licensed under LGPL for Linux UNIX windows and other, many other OSs, without the need for kernel-mode drivers [19]. Its highly portable design makes it favorite for application developers. This can make the communication implemented without going to complicated kernel mode implementations as shown in Figure 15. It also has default support from USB 1.0 to 3.0. It also supports synchronous and asynchronous interface along with all types of transfer types like bulk, interrupt, control and isochronous as shown in Figure 16.

3.3. Interrupt Implementation for Critical Use Cases

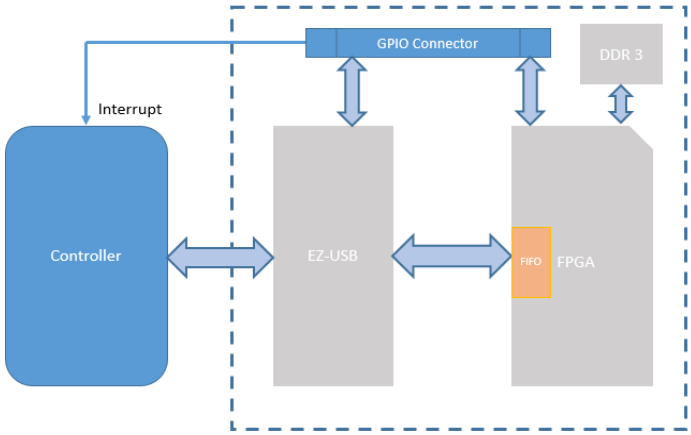

Figure 17. Interrupt implementation for mission citical use cases

FPGA hardware standard code components are added to take care of communication with the host which will make system more portable. USB EZ to FPGA connection part will be fixed so we can reuse the code for any applications. All kind of USB transactions are initiated from host. It demands and over head of port polling to the host application to check if Accelerator has completed the operation. Accelerator has IOs connected to a separate connector from FPGA to address the situation in time critical applications. For such applications it can connect the “Job Done” signal from FPGA will be connected to the host controllers interrupt pin. We will have to modify the host interrupt code to initiate a polling to the shared memory as shown in Figure 17. Even though there is a little portability issue for the design we have added an option for interrupt and other GPIOs in the design to make it more user adaptable.The hardware accelerator had compared with the paper [1], For our design the Artix 7 device function with 96251 logic resources and 131000 memory blocks and can run in 676 MHz system clock.

4. TESTING WITH FFT DESIGNS

The VHDL module calculates two point FFT using the two input samples. The butterfly structure is shown in below figure. This module takes two inputs say ‘a’, ‘b’. The input ‘b’ is multiplied by twiddle factor ‘WN’. The two outputs ‘A’ and ‘B’ are calculated by simple addition and subtraction operation of the two elements as shown in the Figure 18.

Figure 19, shows the state machine of the FFT engine. Data convertor takes data from USB and passes the data to the DDR which can be later taken by FFT engine for calculation DDR interface can be generated using Xilinx core generator tool following image shows DDR interface block for interfacing with FFT engine. The final design can be ported to a pen drive size USB device to plug to any computing system which has a USB interface. The power consumption of the complete reference circuit comes less than 2W which will be enough for a USB device. High level C code will be used to communicate with accelerator from host machine to test the setup. We will be dumping the data through libusb to the DDR for calculation Figure 20 shows the test setup for the hardware.

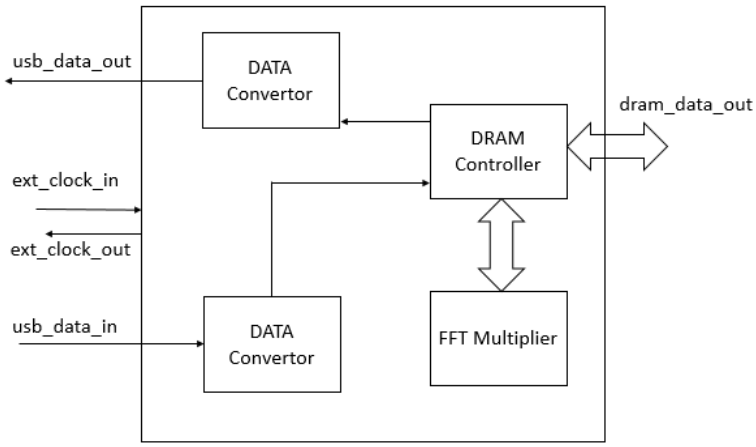

Figure 18. FFT Processor Architecture

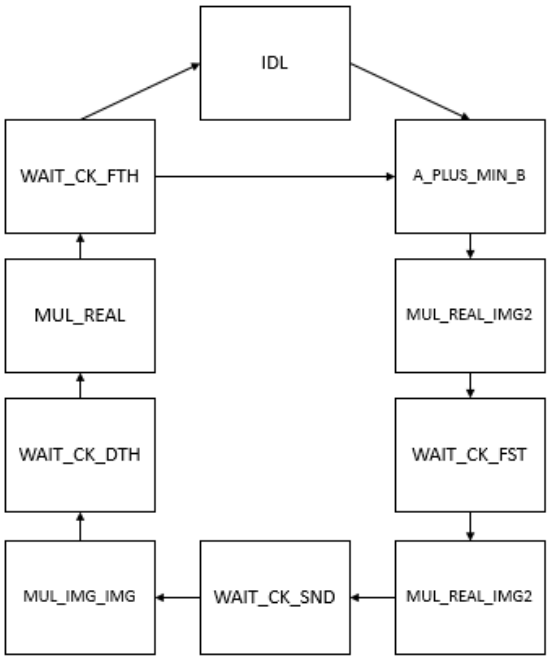

Figure 19. FFT State Machine

3.3. Testing of Hardware Accelerator

We will have to implement functions to access accelerator as wrapper implementation. There are four major functions required for a basic implementation. For reading the shared memory to control the handshake user need a “read_sm” which will read a fixed memory location from hardware accelerator DRAM. While implementing VHDL code we need to isolate this memory location. Same way for passing commands to the hardware accelerator user can use “write_sm”. There are other two functions “read_array” and “write_array” to read the DRAM locations through USB. Address translation for this also will have to be part of VHDL code in FPGA. Function to detect the presence of hardware accelerator and change the program flow is detect_fft_ha following code is used for demonstrating the basic functionality of the total hardware and system software as shown in Figure 20.

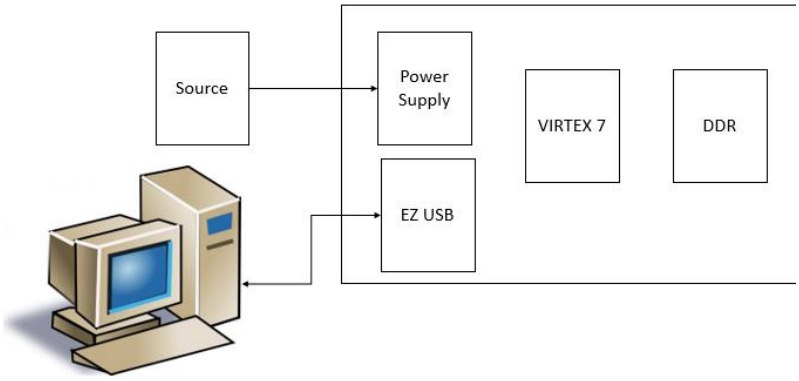

Figure 20. Test setup for FPGA Hardware Accelerator

Below test code can be modified for any kind of application which is less time critical. For more time critical applications, we can use GPIO pin from the FPGA connected to one of the interrupt pin inside that CPU to trigger the result read back. It saves time as well as CPU resource.

Following code chunk shows basic test code for initial testing of setup

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "usb.h" // Header to access installed libusb driver
```

usb.h is header file supplied along with usiblib library for accessing USB hardware.

```
/* fft calculation */

find_fft(*array)
{
    if(detect_fft_ha())
    {
        dump_array(*array,data_in);
        write_sm("0xFF");
        while(read_sm("EE")!=1);//loop untill we g
        {
            res = read_array(data_out);
        }
    }
    else
        res = calculate_using_code(*array);

}
```

4. CONCLUSION AND FUTURE WORK

Investigation has been done on major infrastructures and possibilities of hardware accelerator for embedded application design. Did a detailed literature review of different interface solutions to connect the accelerator to the embedded system and existing studies on its bandwidth and drawbacks. Conducted a study on high level software development environments available in the industry in the area of multi-processing, co-processing and FPGA based solutions. Studied about existing frame works like OpenCL, OpenMP OpenACC etc. Requirements of a hardware design and software components have been discussed to address the problem. Different use cases for a hardware accelerator in the area of encryption compression signal analysis etc and its possibility of using FPGA was analysed and documented. A hardware design has been done using a Xilinx Aritx7 FPGA. Supporting hardware's like DDR interface USB interface power supply designs for different hardware sections etc has been discussed in the project. A test setup design and basic functionality testing process defined.

Complex computing requirements in the area of embedded systems will demand low power, high performance design in the areas where industry is looking for application specific hardware. Development in the area of plug and play hardware accelerators can bring down the system development time drastically. As the silicon manufacturing technology continues to scale downward, the same design can be ported to ultra-small form factor which can help even the product developers to choose from accelerator

modules. Such modular design will help the industry to avoid going through tedious process of safety and other certifications. During the phase of design upgradation. There is big possibility for the design in other computing areas also due to the wide acceptance of the interface used in the design.

Study needs to be conducted to increase the bandwidth by increasing the number of USB channels get bettor through put. Built in kernel / Compiler support and standards will make hardware accelerators more acceptable for designers. FPGA template code need further study to add faster interfaces to the low end embedded platforms like PCI Express / Thunderbolt interface can be added to a raspberry pi board using a multi channel USB Hardware accelerator. Which can be used as glue logic between architectures. Form factor of the design can be reduced to make it acceptable for a system which has space concerns in the existing chassis. Which will help designers to achieve a performance step by adding a small form factor accelerator to the system and change in application code.

REFERENCES

[1] Trio Adiono.et.al, "An SoC Architecture for Real-Time Noise Cancellation System Using Variable PDF Method," *International Journal of Electrical and Computer Engineering (IJECE),* iaes, Vol 5, No.6, December 2015, pp 1336-1346.

[2] Mazin Rejab Khalil and Aseel Thamer Ibrahem, "Design and Implementation of FFT/IFFT System Using Embedded Design Techniquies," *International Journal of Engineering and Innovative Technology (IJEIT)* Volume 3,Issue 6, December 2013. http://www.ijeit.com/Vol%203/Issue%206/IJEIT1412201312_42.pdf (December 2016).

[3] Srihari Cadambi,Igor Durdanovic,Venkata Jakkula,Murugan Sankaradass,Eric Cosatto,Srimat Chakradhar and hans Peter Graf., "A Massively Parallel FPGA-based Coprocessor for Support Vector Machines," *17th IEEE Symposium on Field Programmable Custom Computing Machines,ACM,09 Proceedings of the 2009,*April 2009,pp 115-122.

[4] Mark Reed, "Overview of Research Computing," *ITS Research Computing,* http://www.odum.unc.edu/content/pdf/MReed-ResearchComputingOverview-20121031.pdf (December 2016).

[5] Tomasz S.Czajkowski,Christopher J.Comis,Mohamed Kawokgy Edward S.Rogers "Fast Fourier Transform Implemenataion for high Speed Astrophysics Applications on FPGAs," *University of Toronto,10 Kings College Road,Toronto,Ontario M5S 3G4, Canada.* http://www.eecg.toronto.edu/~czajkow/pubs/ece1373_final_report.pdf (January 2017).

[6] Sanket Dessai and *Krishna Bhushan Vutukuru "Design and Development of Stream Processor Architecture for GPU Using Reconfigurable Computing", International Journal of Reconfigurable and Embedded Systems (IJRES), iaes, Vol 2,No.1, pp 1-14.*

[7] Aaron R. Mandle , "Fast based Hardware Acceleration: A Case Study in Protein Identification," *Division of Engineering,Brown University,BSc Thesis,*2008. http://scale.engin.brown.edu/theses/mandle.pdf (December 2016).

[8] Paulo Possa,David Schaillie and Carlos Valderrama, "FPGA-based Hardware Acceleration:A CPU/Accelerator Interface Exploration," *18th IEEE International Conference on Electronics,Circuits and Systems(ICECS-2011),* 2011.

[9] Roland Doba, "Evolutionary On-line Synthesis of hardware Accelerators for Software Modules in Reconfigurable Embedded Systems," *Faculty of Information Technology,Brno University of Technology, Brno Czech Republic,* https://lis.ei.tum.de/fpl2014/papers/t1c_02.pdf (December 2016).

[10] <n.d.>, "EZ-USBFX3™ SuperSpeed USB 3.0 Peripheral Controller Collatoral Guide," *Cypress Semiconductors,*2015 http://www.cypress.com/applications/ez-usb-fx3-superspeed-usb-30-peripheral-controller-collateral-guide (December 2016).

[11] Grant Martin and Gary Smith, "High-Level Synthesis:Past,Present and Future," *IEEE Design and Test of Computers Volume 26,Issue 4,*July-Aug.2009,pp 18-25.

[12] Valsaraju H. and Vijayakumar G., "Implementing High Speed USB Functionality with FPGA- and ASIC based Design," www.eetimes.com/document.asp?doc_id=127915 (October 2016)

[13] L.Sekanina, "Evolable Hardware," *Hand Book of Natural Computing, Springer Berlin Heidelberg,* 2012.

[14] Rajeshwari Banakar, Stefan Steinke,Bo-Sik Lee,M.Balakrishnan and Peter Marwedel, "Scratchpad memory: A design Alternative for Cache On-Chip memory in Embedded Systems," CODES'02 ACM, 2002.

[15] <n.d.>, "Artix-7 T and XT FPGA Data Sheet," http://www.xilinx.com/support/ (March 2017).

[16] *Cheng Wang,Sunita Chandrasekaran,Barbara Chapman and Jim Holty, "A Portable OpenMP Runtime Library based on MCA APIs for Embedded Systems," UInternational Workshop on Programming Models and Applications for Multicores and manycores(PMAM'13),Proceedings of the 2013,Shenzhen,Guangdong,China, February 2013.*

[17] *D.Pham and et.al, "The Design and Implementation of a First-Generation Cell Processor," in IEEE International Solid-State Circuits Conference 2005., Digest of technical Papers,(ISSCC 2005),February 2005.*

[18] *<n.d.>, "The OpenMP API Specification," www.openmp.org/specifications (March 2017).*

[19] Vinayak Pandit, Sanket Dessai and Shilpa Chaudhari "Development of BSP for ARM9 Evaluation Board", International Journal of Reconfigurable and Embedded Systems (IJRES), iaes, Vol 3,No23,July 2014, pp 62-75.