International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

Performance of Multiprocessor Architecture using Nios II Processor

Tran Hoang Vu
Danang College of Technology,
The University of Danang, Vietnam

***Abstract*- This paper presents the performance of multiprocessor architecture for an embedded system. Firstly, we will design a three-core embedded system using Nios II processors with shared memories on FPGA platform. Secondly, we implement the partition sequential program and Fast Fourier Transform (FFT) on multiprocessor system and measure exactly the execution time of these algorithms. As a result, using multiprocessor, the execution time for sequential program is three times faster and it is also six times faster for FFT algorithm. The result is the evidence to show the performance of multiprocessor architecture on executing programs.**

Keywords- Multiprocessor Architecture, embedded system, Nios II processors , FPGA

I. INTRODUCTION

Embedded Systems has witnessed tremendous growth in the last one decade. Almost all the fast developing sectors like automobile, aeronautics, space, rail, mobile communications, and electronic payment solutions have witnessed increased use of embedded technologies. Greater value to mobility is one of the prominent reasons for the rise and development of embedded technologies. [1]

Embedded system contains processing cores that are either microcontroller or digital signal processors (DSP). A processor is an important unit in the embedded system hardware. It is the heart of the embedded system and has a great impact to system's performance.

Multiprocessor systems possess the benefit of increased performance, but nearly always at the price of significantly increased system complexity. For this reason, the use of multiprocessor systems has historically been limited to workstation and high-end PC computing using a complex method of load-sharing often referred to as symmetric multiprocessing (SMP). While the overhead of SMP is typically too high for most embedded systems, the idea of using multiple processors to perform different tasks and functions on different processors in embedded applications (asymmetrical) is gaining popularity. Altera FPGAs [2] provide an ideal platform for developing a symmetric embedded multiprocessor system,because the hardware can easily be modified and tuned using the SOPC Builder tool [3] to provide optimal system performance. Recent increases in the size of Altera FPGAs make possible system designs with many Nios II processors [4] on a single chip. Furthermore, with a powerful integration tool like SOPC Builder, different system configurations can be designed, built, and evaluated very quickly.

For the reason mentioned above, we design the multiprocessor system using Nios II processor to improve the performance on execution the programs. The main contributions of our work are the following:

- We design the multicore architecture using Nios II processor base on DE2 Altera board.
- We implement the algorithms including sequential caculation and Fast Fourier Transform Algorithm to evaluate the performance of our design.

The rest of the paper is organized as follows. Section II describes hardware design. Section III presents sofware design including algorithms. Section IV describes system design and experimental results. Conclusions are drawn in section V.

II. HARDWARE DESIGN

A. System Block Diagram

In this paper, we design the hardware system as Fig. 1. Our system includes three processors (Nios II/s, Nios II/e and Nios II/f). There are three timers connected to each processor. Every processor has a onchip memory and a shared on chip memory is connected to all three processor. Similarly, a performance counter core is connected to each processor and a system performance counter core that is connected to all three processor. Processor 1 also connected to LEDs and Switches pins. Meanwhile, processor 2 connected to a JTAG module. SDRAM will be used for all three processors to store data. The shared memory will be controlled by three mutex cores and six mailbox cores.

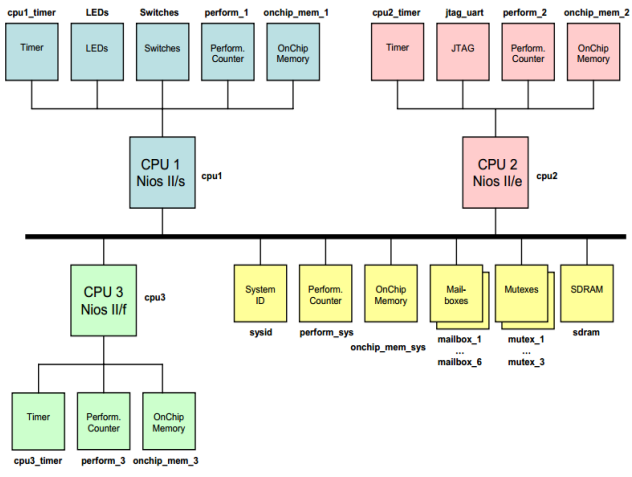

Figure 1. System Block Diagram

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

B. *Nios II Processor Core*

A Nios II processor system is equivalent to a microcontroller or "computer on a chip" that includes a processor, a set of on-chip peripherals, on-chip memory, and interfaces to off-chip memory [4]. Like a microcontroller family, all Nios II processor systems use a consistent instruction set and programming model. The Nios II architecture describes an instruction set architecture (ISA). The ISA in turn necessitates a set of functional units that implement the instructions. A Nios II processor core is a hardware design that implements the Nios II instruction set and supports the functional units described in this document (Fig 2). The processor core does not include peripherals or the connection logic to the outside world. It includes only the circuits required to implement the Nios II architecture.

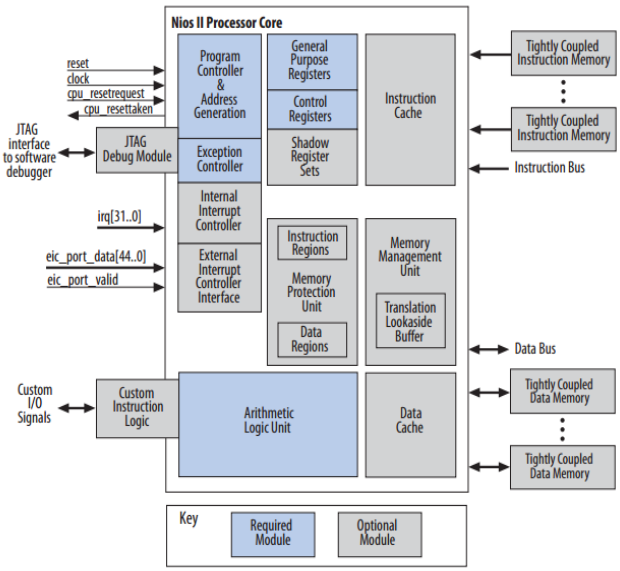

Figure 2. Nios II Processor Core Block Diagram [4]

C. *Shared Memory*

Shared memory can be used for anything from a simple flag whose purpose is to communicate status between processors, to complex data structures that are collectively computed by many processors simultaneously as Fig 3.

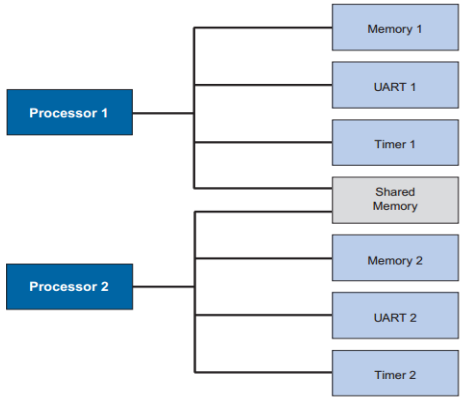

Figure 3. Multiprocessor System with Shared Memory [5]

D. *SDRAM Controller Core*

The SDRAM controller core with Avalon interface provides an Avalon Memory-Mapped (Avalon-MM) interface to off-chip SDRAM [6]. The SDRAM controller allows designers to create custom systems in an Altera device that connect easily to SDRAM chips.

SDRAM is commonly used in cost-sensitive pplications requiring large amounts of volatile memory. While SDRAM is relatively inexpensive, control logic is required to perform refresh operations, open-row management, and other delays and command sequences. The SDRAM controller connects to one or more SDRAM chips, and handles all SDRAM protocol requirements. Internal to the device, the core presents an Avalon-MM slave port that appears as linear memory (flat address space) to Avalon-MM master peripherals.

The core can access SDRAM subsystems with various data widths (8, 16, 32, or 64 bits), various memory sizes, and multiple chip selects. The Avalon-MM interface is latency-aware, allowing read transfers to be pipelined. [7].

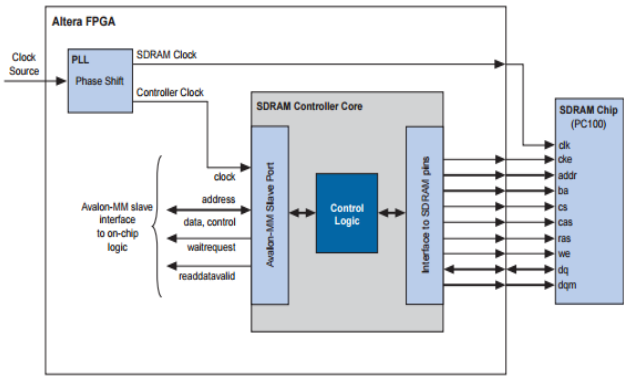

Figure 4. SDRAM Controller with Avalon Interface Block Diagram [7]

Fig. 4 shows a block diagram of the SDRAM controller core connected to an external SDRAM chip. SDRAM will be used to store code section of all three CPUs. Therefore this core connects to both Data Master and Instruction Master for all three CPUs.

III. SOFTWARE DESIGN

This section will present about the performance of multicore system in comparison with single core system.

A. *Acceleration*

In this section, we start from the file *sequential.c*. For parallel implementation purpose, we have to rewrite from two recursive functions series1 and series 2 to two corresponding normal functions using for loop. Each CPU in multiprocessor system will execute a different part of these loops.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

- Series 1

```
int series1(int y, int x)
{
        int result = 0;
        int i;
        for (i = y; i <= x; i++)
        {
                result = result + 2*i;
        }
        result = result - 1;
        return result;
}
```

- Series 2

```
int series2(int y, int x)
{
        int result = 0;
        int i;
        for (i = y; i <= x; i++)
        {
                result = result + i*i;
        }
        return result;
}
```

➢ Parallel Implementation 1

This program will run and calculate the total execution time of function series1 and series2 for each case of k by single core. Fig 5 shows the flow chart of the parallel implementation.

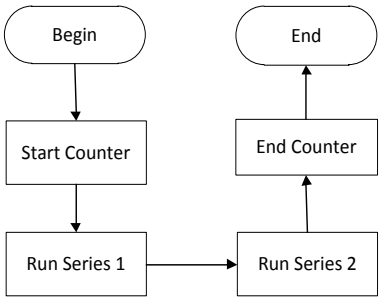

Figure 5. Paralled Implementation 1 Flow Chart

➢ Parallel Implementation 2

In order to reduce the execution time of the program, we will select two candidates implementation. CPU2 and CPU3 will execute these function concurrently, each CPU will do half of for loop as Fig 6.

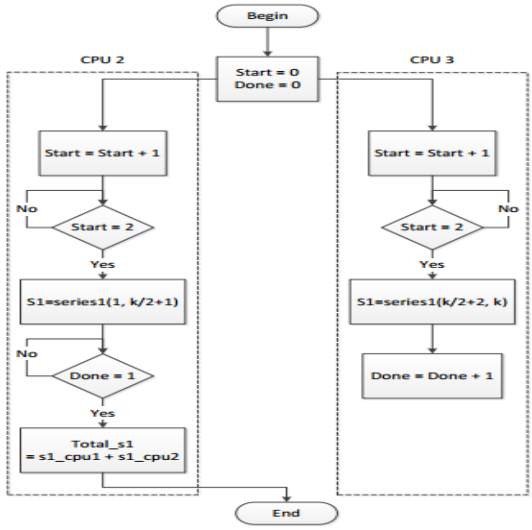

Figure 6. Parallel Implementation 2 Flow Chart

This program will be executed by two processors at the same time, therefore if for loop has to execute k times, CPU 2 will execute the loop from 1 to k/2 + 1, and CPU 3 will execute the remaining loop, from k/2 + 2 to k.

➢ Parallel Implementation 3

In this sector, we expand the program so that it can be executed parallel by three CPUs, each CPU will execute one-third of for loop as Fig.7.

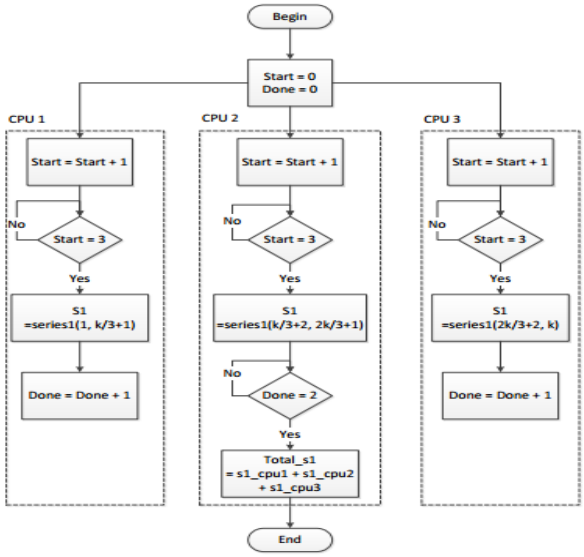

Figure 7. Parallel Implementation 3 Flow Chart

Similar to the program that using two cores, if for loop executes k times, CPU 1 will perform the loop from 1 to k/3 + 1, CPU 2 will perform the loop from k/3+ 2 to 2k/3 + 1, and CPU 3 will perform the remain loop, from 2k/3 + 2 to k.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

B. *Fast Fourier Transform Algorithm*

We will use multiprocessor system to calculated 64-point Fast Fourier Transform. We will use Cooley-Turkey Radix-2 algorithm [8][9] to calculate 64-point Fast Fourier Transform [10].

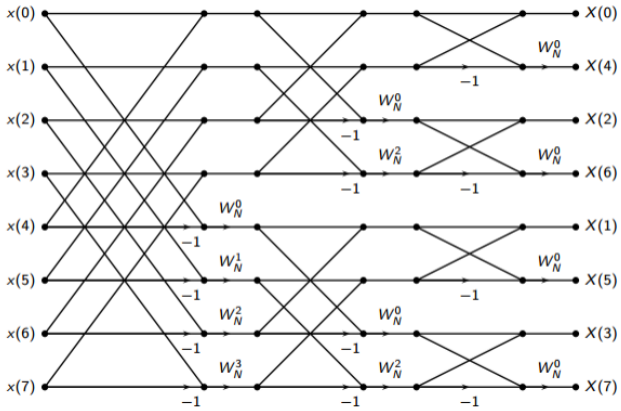

Figure 8. The example 8-point FFT using Radix-2 diagram

Fig 8. shown the example 8-point FFT using Radix-2 diagram. 8-Point FFT using Radix-2 has to be calculated in three stages. In our case, we need six stages to calculated 64-point FFT and another stage to re-order results. For each stage (except reorder stage), we use for loops to calculate the result follow the diagram. Fig.9 shown the flow chart of stage 1.

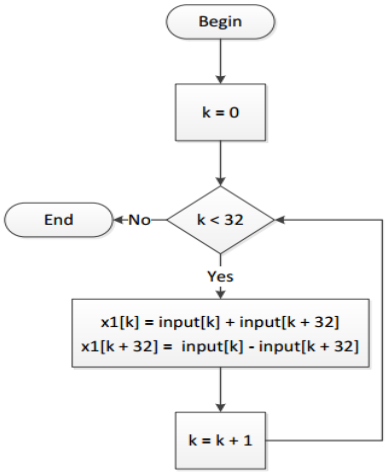

Figure 9. Radix-2 64-point FFT - Stage 1

Using multipcore, in each stage, for loops are independent (Fig. 10). Therefore we can divide these for loops into two other loops. CPU2 will be used for synchronization and for I/O operations, while CPU 1

and CPU 3 are used for calculate the FFT. CPU 1 and CPU 3 start the calculation at the same time and they execute the same for loop but on different data. After they finish the calculation, they signal to CPU 2.

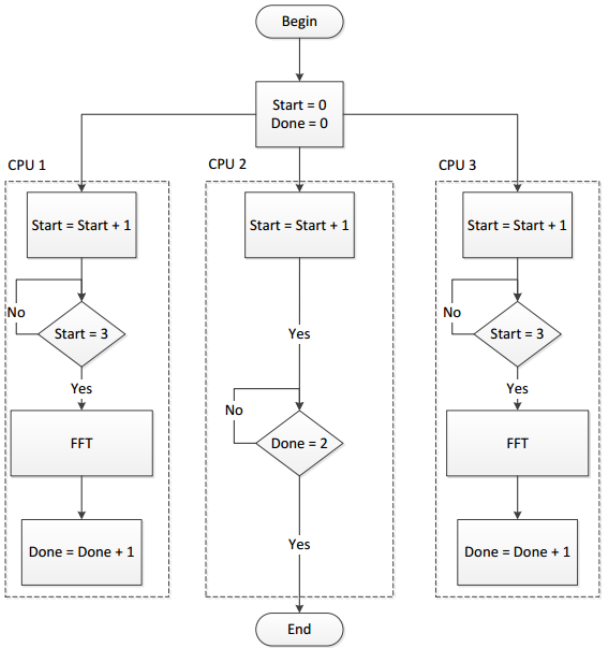

Figure 10. Radix-2 64-point FFT Algorithm using Multicore.

IV. EXPERIMENTAL RESULTS

A. *Multipcore System Design*

Our hardware system is built by SoPC builder tool [3] based on DE2 Altera board [11]. Table. 1 shown the size of our design. Our system possesses 45% in total logic elements of DE2 board, and maximum clock frequency is about 61.16 MHz.

TABLE 1. HARDWARE DESIGN SIZE SUMMARY

	Used	Available	Percentage
Logic elements	15102	33216	45%
Registers	8177		
Pins	56	475	12%
Memory bits	184960	483840	38%
Embedded multiplier elements	0	70	0%
Maximum clock frequency	61.16 MHz		

B. *Measuremental Results for Acceleration*

In the Parallel Implementation 1, only CPU 2 execute these function and calculate the performance time using API functions of performance counter core.

In the Parallel Implementation 2 and 3, to get the exactly result of execution time, it is important that CPU1, CPU 2 and CPU 3 start calculate these functions at the same time. Shared variable *start* will be used for that. After CPU 3 finishes each function series1 and series2, they store result in shared memory and increment the *done* variable, then CPU 2 will calculate the total result. We set 9 diffirent values of k and run with four implementations including Sequential, Parallel Implementation 1, 2 and 3. The results compares in the Table. 2.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 06, June - 2017

TABLE 2. ACCELERATION RESULTS (S)

k	Sequential Implementation	Parallel Implementation 1	Parallel Implementation 2	Parallel Implementation 3
1	0.00005	0.00009	0.00057	0.00072
5	0.00030	0.00025	0.00067	0.00090
10	0.00064	0.00064	0.00087	0.00102
25	0.00203	0.00138	0.00157	0.00162
50	0.00425	0.00305	0.00273	0.00247
100	0.00907	0.00660	0.00512	0.00380
250	0.02423	0.01780	0.01316	0.00843
500	0.08318	0.03794	0.02783	0.01728
1000	0.10700	0.09696	0.05884	0.03492

From Table 2, if the value of k is small, the program will perform faster using single core. Because the time need to accessing shared memory instructions and waiting time takes a considering amount of time in comparison with the time need to execute series1 and series2 functions. However, if k is great, the program is faster if using more mulprocessor. If we use the parallel implementation 3, the execution time is three times faster compared to sequential implementation.

C. Measuremental Results for FFT

In order to calculate the executed time of this algorithm, we use performance counter core. Firstly, we run FFT algorithm on CPU1 and capture the executed time. Fig. 11 shows the executed time that performance counter calculated.

```
--Performance Counter Report--
Total Time: 8.91456 seconds  (445727793 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|FFT            |  100|    8.91455|      445727597|          1|
+---------------+-----+-----------+---------------+-----------+
```

Figure 11. The result for Fast Fourierm Transform using single processor

As the result, using single processor needs about 9 seconds to execute 64-point Fast Fourierm Transform using Radix-2 algorithm.

Using multipcore, both CPU 1 and CPU 3 have to start calculation of each stage at the same time. Shared variable start will be used for that. Initially *start* variable equals zero, each CPU have to increment this variable before they start the functions.

When a processor calculate completely one stage, it will increment *done* variable and wait for the increment of the other CPU. Fig. 12 shows the execution time that performance counter calculated.

```
--Performance Counter Report--
Total Time: 1.4153 seconds  (70764935 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|FFT            |  100|    1.41529|       70764627|          1|
+---------------+-----+-----------+---------------+-----------+
```

Figure 12. The result for Fast Fourierm Transform using multiprocessor

Interestingly, the execution time decreases significantly from about 9 seconds to 1.5 seconds when we use three cores.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we design successfully the multiprocessor architecture using Nios II processor. We also execute the sequential program and FFT algorithm on single core and multicore in order to compare the performance of our system. As the result, the execution time of these algorithm reduces considerably when we use the multiprocessor architecture.

In the future, we will research and propose the multiprocessor architecture for image processing technology using FPGAs that could decrease significantly the execution time compared to use single processor.

REFERENCES

[1] http://www.vectorindia.org/applications_of_embedded_systems.html
[2] https://www.altera.com/products/fpga/overview.html
[3] SOPC Builder User Guide, December 2010 Altera Corporation.
[4] Nios II Classic Processor Reference Guide, 2016.06.17 Altera Corporation.
[5] Creating Multiprocessor Nios II Systems Tutorial, June 2011, Altera Corporation
[6] Using the SDRAM Memory on Altera's DE2 Board with VHDL Design, Altera Corporation.
[7] Embedded Peripherals IP User Guide, 2016.06.17, Altera Corporation.
[8] https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm
[9] The Discrete Fourier Transform, Part 2: Radix 2 FFT By Douglas Lyon, Vol. 8, No. 5, July-August 2009.
[10] Fourier Transforms and theFast Fourier Transform (FFT) Algorithm Paul Heckbert, Feb. 1995, Revised 27 Jan. 1998, Notes 3, Computer Graphics 2, 15-463.
[11] DE2 Development and Education Board User Manual, 2006 Altera Corporation.