International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 1 No: 4 June 2016

Performance optimization on System on Chip using I²C Concept 6G Computer

D.Kanimozhi[1], P.Karthikeyan[2], S.Kirthika[3]
[1](Assistant Professor, ECE), M.Tech, [2,3](Student), B.E (ECE)
[1,2,3]C.K.College of engineering and technology, Cuddalore,INDIA.
[1]kanimozhi.gk@gmail.com, [2]karthikeyanperumalece@gmail.com

Abstract: In this paper we deliberated the delay reduction in multi core processor on soc to perform multi tasking by means of I²C concept and controlled cache memory. To embed multiple applications on the CHIP, control the process execution. In that we are using EHCI (Embedded Host Controller interface) to communicate outside peripherals to chip. The special concept disused dual mode of operation active and sleep mode .To execute overall system operation by means of embedded OS.

Keywords: Cache , EHCI , I²C, SPARTEN 3E Kit, Xilinx 14.7 ISE Design Suite.

1. INTRODUCTION

The aim of the system is to reduce the delay on the process execution by using System on chip concept. To build four processor in asymmetric technique, Complex instruction set computer, reduced instruction set computer and digital signal processor are used. Thread level penalization method is used, to utilize the memory space of chip. I²C concept introduces two modes of operation active mode and sleep mode. The system interfaces multiple applications such as automation, wireless sensor networks, IOT and general purpose embedded operation. Special embedded OS system is used to control and monitor multiple applications. In order to implement the system on soc with the help of Verilog coding in Xilinx 14.7 design suit software and also can be implemented with the help of Spartan 3e kit.

2. THE CHOSEN COMPUTATIONAL PROBLEM

The chosen problem was to compute using an asymmetric technique. The general idea of asymmetric block is as follows: multiple source data is first split into separate blocks of fixed and equal length.

The above technique describes how the computed block works with one data block. There are a few ways this behaviour can be extended to multiple blocks. The one that was chosen for this project is called first in first out technique. In this method, each data block is processed separately. The data blocks are treated in a fully-independent manner, and therefore, this mode is well-suited for designing a parallel algorithm.

3. PROCESSOR DESIGN AND RCHITECTURE

In this section, the architecture of the processor and its interface is first presented. After that, the scalability of the proposed architecture is discussed.

A. Interface

The processor has its own simple instruction set. It allows the building of computing programs which operate on different amounts of data.

It has to be done by directly inputting the binary content into the Verilog design file. Still, the results multiple data processing can be monitored on a set of appropriate outputs, like LEDs.

It monitors and controls the temperature in surroundings when using temperature sensor. Changes are indicated via output device.

This can be upgraded by adding support for the desired memory or interface type to the design, thus allowing data input and output to be independent from PLD programming files.

B. Schematics

The general architecture of the processor is presented in Figure1. The processor core consists of four main modules, called A, B, C, D, E and F. Additionally, there is also a monitoring module which interprets user-control signals and sends back their results via a specified output.

The monitoring module allows the switching of the output signals indicators between different core modules, which are especially useful when the number of outputs is limited. This module is also able to trap execution of the program, count the elapsed clock ticks, and detect idle statuses of particular modules.

Therefore, it is able to determine the end of processing and measure elapsed time automatically.

The embedded host control interface (EHCI) (labelled A) can directs the outside peripherals to the respective processor. It optimizes the overall process in a controlled manner.

The sequential I^2C processor (labelled B) is responsible for the interpretation and execution of a user-defined program. It executes all instructions of the program on demand.

The queuing module (labelled C) is responsible for scheduling tasks received from the module B for the D processors. A new task is scheduled for the first idle B processor. If all D's are busy, the task is put into an internal FIFO queue. When the queue becomes full, further tasks are not accepted until a place in the queue is freed.

After that, it starts processing the task. The processor communicates with the cache and memory module, requests the needed data, combines them, and stores the results back in the memory.

The cache memory L1 (labelled E) is the most-passive module. It holds the data and key for processing and allows for concurrent access from all D modules. According to the block queuing idea, the parts of data on which the processors D operate should be separate, although some safety mechanisms still have to exist in the design for concurrent read/write problems. The priority solution was chosen to minimize unnecessary overheads.

The parallel memory can store the data and valid instructions. It drives data to cache memory to process the data in the system.

C. Scalability

The processor architecture can be scaled. The main factor which determines the speed of the processing is the number of implemented D modules. The D modules can be treated as the actual cores of the processor. Of course, there has to be proper support for them from the adjoining A, B, C and E modules.

It mostly means additional ports and buses, and rescaling of the internal algorithms, which can be done by a modification of the design files. Module A is - according to its name EHCI makes interrupt for interface to the system. Module B is - according to its name I^2C sequential and not meant to be scaled. Module C is – according to its name queue and its way to process the data on the system.

In module E is – according to its name L1 cache and it has execution EX1and execution EX2 with cache control can drives the data effectively to the system.

4. PARALLEL CHARACTERISTICS

In the system Moore's model was used for parallelization to reduce the complicity and the function with the help of Amdahl's law.

In order to determine the quality and correctness of the design, processor performance had to be measured. The most basic metric that can be measured is the amount of time needed to complete the processing, depending on data size and number of cores used. From that, the standard parallel computing characteristics can be derived. The correctness of the processing was verified fairly easily. All that was needed was to run the program twice and check whether the memory content first shifts to an encrypted form and then back to its normal shape.

Initial tests showed that a little-coarser granularity is required to make full use of all 4 cores. That is why both setup and result sections are split in two parts and cover both the initial and final tests.

A. Experimental setup

For test purposes, both program for module A as well as sample data and key stored in module F, had to be prepared. Additionally, all the adjustable parameters of the processor had to be determined.

International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 1 No: 4 June 2016

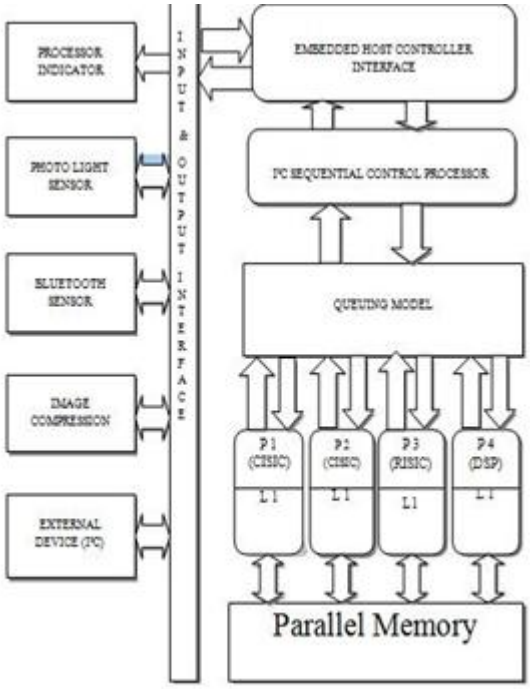

Figure 1. Architecture of multi-core processor in SOC

The general test configuration is as follows:
• there are at most 4 parallel processors available (p = 1, . . , 4),
•module B internal queue has 4 places
(which is more than enough),
• the memory consists of 32-bit words.

5. RESULTS

The results cover all possible combinations of problem size n and number of processors p. The time $T(n, p)$ was measured in clock ticks automatically by the monitoring module, so it is exact and independent of clock frequency.
For each set of parallel characteristics, two charts are provided: one with a standard view: $f(p)$ and n-dependent data families and an alternate view: $f(n)$ and p-dependent data families.
Two main characteristics were taken for each case

- speedup $S(n, p) = T(n,1) / T(n,p)$
- efficiency $E(n, p) = S(n,p)/p$

A. Finer granularity

The processing times for all cases are shown in Table 1. The values are ordered by the indices p and n. Always $T(n1, p1) \rightarrow T(n2, p2)$ where $n1 \rightarrow n2$, $p1 - p2$. So, the more processors used or the smaller the problem size, the shorter the processing time. It proves the correctness of the design on a very basic level.

Table 1. Elapsed time – finer granularity

p/n	1	2	3	4	5	6	7	8
1	39	64	89	114	139	164	189	214
2	39	48	64	73	89	98	114	123
3	39	48	57	66	75	84	93	102
4	39	48	57	66	75	84	93	102

For n = 1, the values are all the same no matter how many processors are used. By the time the fourth task is generated, one of the Cs finishes its job and accepts the new task, and so the fourth C processor is idle all the time. To make real use of 4 cores, the task size should be bigger or module A should work faster.

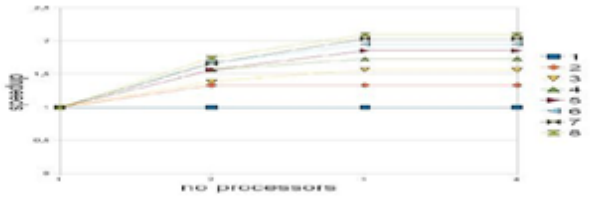

Figure 2. Speedup – finer granularity (standard view) .

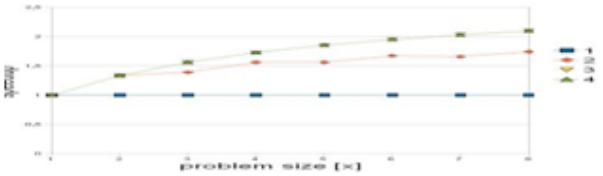

Figure 3. Speedup – finer granularity (alternate view).

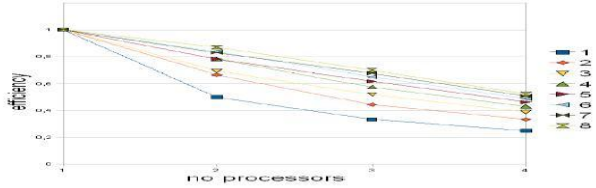

Figure 4. Efficiency – finer granularity (standard view).

International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 1 No: 4 June 2016

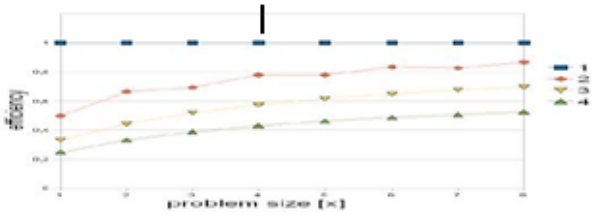

Figure 5. Efficiency – finer granularity (alternate view).

Figure 5 shows that efficiency rise along with the problem size. It means that the sequential part must decrease at the same time. In this architecture, the sequential part can be defined as the time when at least one C unit is idle, despite the fact that there are still tasks that need to be processed The B and D modules are working fast enough not to produce any delays, so the only source of possible delays in the design is the sequential processor B.

B. Coarser granularity

Processing times for all cases are shown in Table 2. The results are quite similar but this time, the fourth core really makes a difference.
The increased granularity gave module A, a chance to occupy all of the C cores, while working at the same speed.

Table 2. Elapsed time - Coarser granularity.

p / n	1	2	3	45	6	7	8
1	63	112	161	210259	308	357	406
2	63	72	112	121161	170	210	219
3	63	72	81	112121	130	161	170
4	63	72	81	90112	121	130	139

It can also be noted that, for $n = 4$, the values are slightly better than in the corresponding finer-granularity case with $n = 8$. Figures 6 and 7 show the speedup characteristics. The corresponding finer granularity chart presented in Figure 2 had no crossed lines. Here in Figure 6, it can be seen that speedup for $p = 2$ is better for $n = 2$ rather than 3. The same situation is for $p = 3$ and $n = 3, 4$, or $p = 4$ and $n = 4, 5$. Also, the alternate view presented in Figure7 now has a more-distinct shape. Any other values of n are less optimal and cause speedup and efficiency drops.
Figures 8 and 9 present efficiency characteristics. Similar to speedup, the peaks are located at full multiples n of p values. Also, the bigger the problem size, the more the distortions are scaled down; but, they still have the same shape.

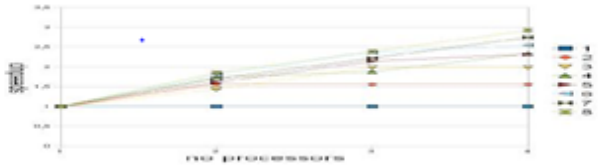

Figure 6. Speedup – coarser granularity (standard

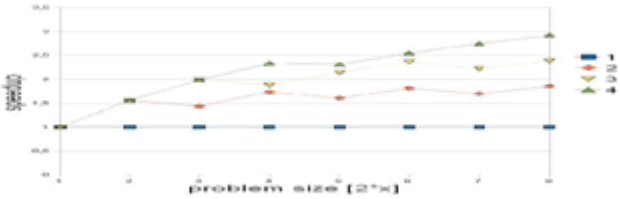

Figure 7. Speedup – coarser granularity (alternate view).

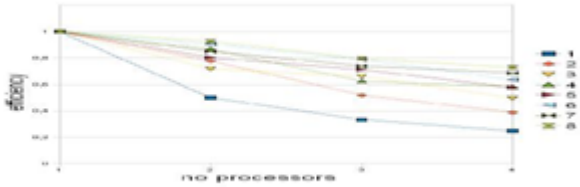

Figure 8. Efficiency – coarser granularity (standard view).

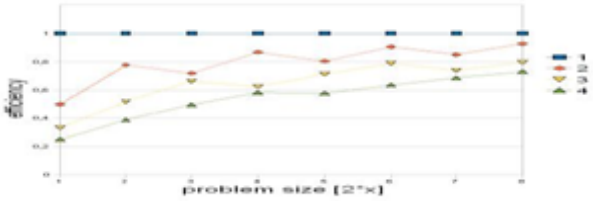

Figure 9. Efficiency – coarser granularity (alternate view).

C. Corollaries

The time measurement is strict — it is not measured in real time units but in the number of processing steps. Also, the processor itself works in a determined way; and, for a given task, it always needs the same number of steps to complete

it. Therefore, the measured values should not be random. Indeed, a closer look reveals that all of them can be described using a single formula:

$T(n, p) = \{di + y + dl(n - 1) + (y - dl \cdot p)[n-1/p]$ if $y > dl \cdot p$
$di + y + dl(n - 1)$ otherwise (1)

where:

di – initial delay [clock ticks] – total processing time before the first command actually starts being executed by one of the C processors;

dl –loop delay [clock ticks] – delay between generating subsequent commands by the A processor (processing time of all the instructions inside the loop1 of the program);

y –time needed to compute one data block consisting of *x* memory words — it depends on a chosen algorithm and its implementation in module C.

The additional part $(y - dl \cdot p)\ [n-1/\ p]$ describes the fact that, when the task size is too big (or, conversely, there are too few processors), some tasks will have to wait until they are generated before they will actually be processed. When $y < dl \cdot p$, it means that there is always a free C module to handle a new task, and the speed of the processing is only limited by the speed of module A. The *y* element in $di + y + dl\ (n - 1)$ corresponds to the
processing time of the last task, after module A has finished its job.

The factor $[n-1/\ p\ ]$ is an integer division. When $y \geq dl \cdot p$, each *p*-th task has to wait $y - dl \cdot p$ cycles in queue before being processed. The first task that has to wait is $(p + 1)$-th task, followed by $(2p + 1)$-th, $(3p + 1)$-th, and so forth. Of course, the formula (1) is only true for a user-defined program.

The formula (1) is true for $p = 1$, where the C processor is always busy. It is also satisfied for both considered granularities even though the finer granularity does not use fourth C processor at all. Based on that, the formula should also be true for any valid range of parameters *n*, *p*, *x*, *y*, but it was not tested in this experiment.

6. CONCLUSION

The project had two main goals to accomplish: first, design a microprocessor multi core architecture which would show the significant improvement over sequential computing; and second, make this architecture scalable.

The achieved results did show an improvement with speedup reaching 1.85 with two cores and 2.92 with four cores active. This yields the efficiency of 0.93 and 0.73 respectively. Additionally, the sequential part decreases with the problem size, so the characteristics should be even better with more tasks to process.

The architecture is also scalable. It had already been rescaled from 2 cores to 4 in its current shape, so further expansion seems limited only by the number of necessary repeatable changes in the design files. But this still could be managed with the help of appropriate Verilog commands.

REFERENCES

[1] R. Kumar et al., "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction," Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture, IEEE CS Press, 2003, pp. 81-92.

[2] Nitin Chaturvedi1 S Gurunarayanan2: study of various factors affecting performance of multi-core processors, International Journal of Distributed and Parallel Systems, (IJDPS) Vol.4, No.4, July 2013.

[3] Manoj Kumar Gouda, D.Yugandhar: Design of Multi-core Processor using Multithreading Technique, Blue Eyes Intelligence Engineering & Sciences Publication Pvt. Ltd, October 2014.

[4] Christian Märtin, Multi-core Processors: Challenges, Opportunities, Emerging Trends, embedded world Conference 2014.

[5] Verilog Hardware Description Language Reference Manual, http://ecad.tu-sofia.bg/soc/data/verilog/verilog.pdf

[6] Digital Circuit Design Xilinx ISE Tools, http://www.Xilinx.com/products/software/products/ Xilinx ISE Tools -index.html.

