Integrating MRPSOC with multigrain parallelism for improvement of performance

[1]Swathi S T, [2]Kavitha V
[1]PG Student [VLSI], Dept. of ECE, CMRIT, Bangalore, Karnataka, India
[2]Ph.D Scholar, Jain University, Bangalore, Karnataka, India

Abstract **- There are several techniques to reconfigure the instruction set processors. One such technique is Multi Reconfigurable Instruction Set Processor System on Chip (MRPSOC). Integrating MRPSOC with multigrain parallelism for improvement of performance is done in this paper. By using MRPSOC, performance of the system is increased. Multimedia application computing can be accelerated by using multigrain parallelism. By implementing this integrated processor extra feature can be added to MRPSOC. Multiple data is packed in a single register which forms a vector; this vector of multiple data is fetched to MRPSOC at a time. Since MRPSOC is a combination of MPSOC and RISP processor, instruction level parallelism can be implemented in MRPSOC. After execution of operations in MRPSOC the multiple outputs can be stored at different memory locations of a same memory system simultaneously. This thesis is aimed to design MRPSOC interfaced with data level, instruction level and memory transfer level parallelism. Form this paper it is concluded that by using both MRPSOC and multigrain parallelism in common platform high speed computation can be achieved.**

Index terms **- Reconfigurable Instruction set Processors (RISP), Multigrain parallelism , Multi Reconfigurable Instruction set processor system on chip(MRPSOC)**

I. INTRODUCTION

General purpose computers like PCs, workstations the idea for designing the general purpose processors came into picture. Computational speed of general purpose processors is high but the cost is higher when compared to DSPs as well as micro controllers. All the techniques which are involved in the improvement of CPU speed are applied to GPP's. Higher performance computation cannot be done by general purpose processors so ASIC's are designed which are energy efficient supports for high performance computing. Due to shrinkage of geometry in today's criteria it is very difficult to design an ASIC. For case of sub -100nm designs, the manufacturing costs of ASICs are rising day by day. Non recurring design as well as manufacturing costs of ASIC results in heavy breakage of fixed amount unit costs. So to overcome all the problems mentioned above a new programmable platform ASIP's come into picture. The programmability of ASIP supports for vast amount of multiple related applications and also various generation of any application is able to map the same ASIP.

RISP's are similar to ASIP's, here the most repetitive as well as time consuming parts of an application are run on dynamic, adoptive functional unit called as RFU (Reconfigurable functional unit). A RISP mainly contains a microprocessor core and reconfigurable logic. RISP is of same type as that of ASIP only difference between them is ASIP has specialized functional units, Whereas RISP have reconfigurable functional units. MPSOC is nothing but multiprocessor system on chip. MPSOC has risen over decade as very large scale integration system. In case of MPSOC more than two multiprocessors are integrated on a single chip. This MPSOC is a VLSI system which includes most of the components which are required for an application. Mainly these applications come under embedded applications MPSOC symbolize a foremost and well defined sector of multiprocessors.

By using RISP and MPSOC in a single platform a better results can be achieved. The combination of RISP and MPSOC results in MRPSOC". Completion time can be reduced at two levels in case of MRPSOC. At the starting level, programs are executed simultaneously on the parallel processing elements. At the next level, the time taken for execution can be minimized with the help of running most complex sections of the identified task on RFU unit.

By integrating MRPSOC and multigrain parallelism the completion time can be reduced. In this proposed platform data level parallelism, instruction level parallelism and memory transfer level parallelism is integrated with MRPSOC. Normally for MRPSOC only two inputs can be given at a time but if data level parallelism is included, it is possible to give more than two inputs, so execution speed will be increased. By including memory transfer level parallelism, at a time more number of resultant data of MRPSOC can be stored in different memory location of same memory system. MRPSOC processor itself includes instruction level parallelism since MRPSOC has RFU unit. So this enhanced processor is used for multimedia applications as well as high performance applications.

II.PROPOSED SYSTEM

To improve the performance and to reduce the compilation time MRPSOC is integrating with multigrain parallelism (data, instruction and memory transfer level parallelism). The fig 1 describes the block diagram of proposed system. Data level parallelism is included to fetch the data to MRPSOC; multiple data's can be fetched at a time. MRPSOC processor is the main block of the system where all the operations are done.

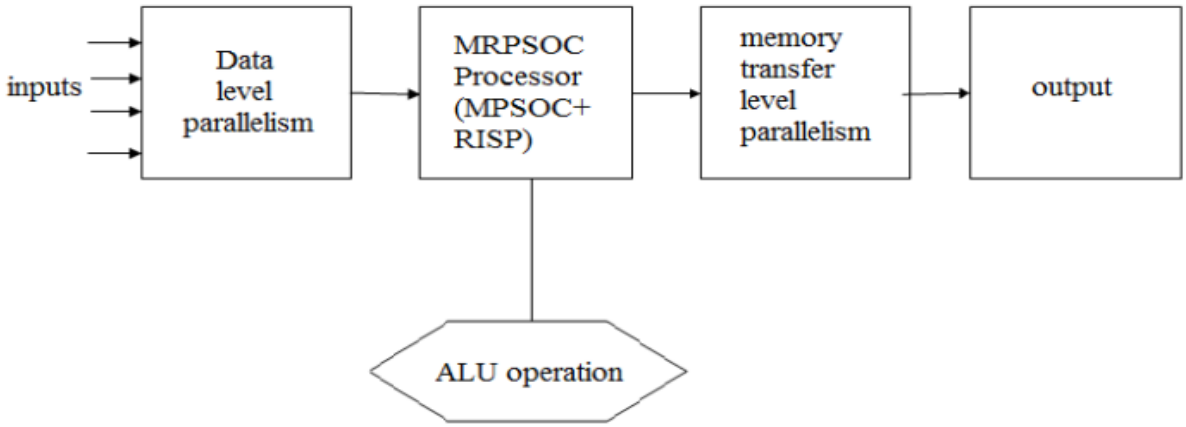

Fig 1: block diagram of integrated MRPSOC and multigrain parallelism

MRPSOC is the combination of MPSOC and RISP. RISP contains RFU units. Performance of the system is improved by using MRPSOC, since complex sections of program can be run on RFU unit and execution of program can be done simultaneously on ALU and RFU. ALU operations will be done in MRPSOC. Memory transfer level parallelism is mainly used to store the resultant data in parallel. To store the result simultaneously, multiple numbers of memory transfer requests should be arranged in parallel before computing. To transfer the data between main memory and local storage DMA is used.

A. RECONFIGURABLE INSTRUCTION SET PROCESSOR

With the help of specially designed hardware reconfigurable instruction set processor is having capacity to adopt instruction set depending on the application which is being executed. Both the software flexibility and hardware efficiency is combined in RISPs. To overcome the problems which are seen in ASIP's, RISP came into picture. RISP's are similar to ASIP's, here the most repetitive as well as time consuming parts of an application are run on dynamic, adoptive functional unit called as RFU (Reconfigurable functional unit).

A RISP mainly contains a microprocessor core and reconfigurable logic. RISP is of same type as that of ASIP only difference between them is ASIP has specialized functional units, Whereas RISP have reconfigurable functional units. Processor core of RISP supplies programmability and reconfigurable functional unit adopts specific application to the processor. Interfacing reconfigurable logic with base processor includes related issues like data transfers to and from the reconfigurable logic, and also co-ordination between the two elements. Design of reconfigurable logic should take care of issues like RFU granularity, interconnection between logic and processor. RPU and Microprocessor can configurable to work at the same time in different tasks. Any of the application can be accelerated by using this configuration. The need for external glue logic can be eliminated here.

The performance of the micro processor is affected by the position of the RFU.

- Attached processor: on the I/O reconfigurable logic is placed.
- Coprocessor: beside or next to the processor the logic is placed.
- In functional processor the logic is placed within the processor.

In instruction types there are two types: stream based instruction and custom instruction. In stream based instruction large number of data is processed in blocks manner whereas for custom instruction for small amount data is processed. In various ways instruction types can be divided. With the special opcode instruction coding is done. Memory access in reconfigurable instruction is through memory ports. Granularity of RFU can be divided into fine gain and coarse grain. Fine grain is more flexible and memory configurable is also large compared to coarse grain. To get required functionality interconnection is done in two ways intra segment interconnect and extra segment interconnect. Depending on the size of configurable data the reconfigurable functional unit works.

B. Architecture of RFU:

Especially the design of RFU consumes large area. In this design efforts are made to decrease the size of RFU without affecting the performance. In the path of processing data after the ALU unit RFU structure is placed mainly to transfer the data at a faster rate. Multiple functional units (FU's) are included in RFU structure which can be organized as matrix. The granularity of RFU is also decided which may be coarse grain accelerators or fine grain reconfigurable accelerators which depends on the applications.

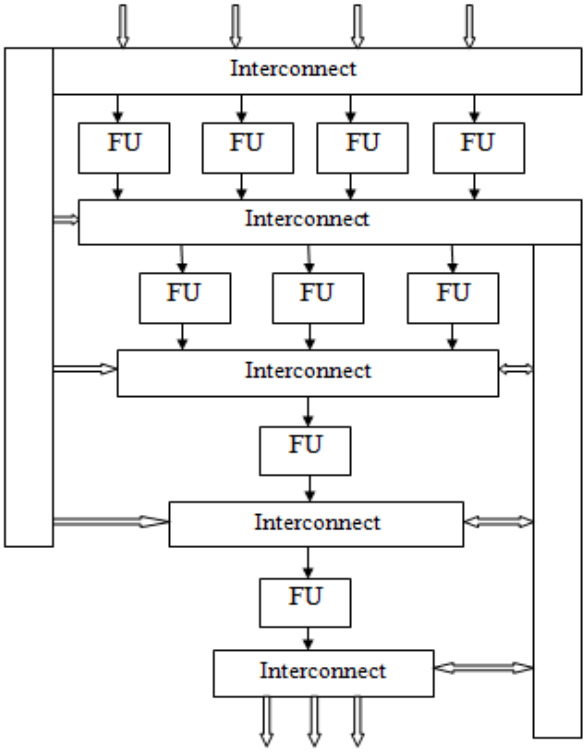

Fig 2: RFU structure [1]

Except the instructions like load, store, multiplication and division the RFU supports almost all the MIPS instructions. To the file register the input data is given in RFU then resultant is written back to the file register itself. Two types of memory configuration are provided by designed RFU unit, that is, function unit's operation type and its interconnection method. The amount of CI's which can be included in the RFU decides the size of corresponding configuration memory bits.

To decrease the implementation overhead the RFU is designed by considering several constraints. Input/output numbers are identified at the beginning. For custom instruction four inputs, three outputs are considered as shown in fig 4.2. Arrangement is done in such a way that in each RFU level number of function units is placed. The proposed RFU contains four inputs and three outputs and four levels of RFU, at each level the FU's are arranged in decrementing order (four, three, two, and one). The RFU level decides the clock so this proposed structure of RFU has three clocks to finish the operations. These FU units designed communicate with the help of programmable multiplexers. To reduce configuration bits as well as interconnections clustered FU's are used which does not affect the performance.

C . Methodology for MRPSOC:

Methodology for MRPSOC is described in two steps. In first step, processing elements are executed in parallel. In next step, complex parts of the task are run on RFU unit. Methodology for MRPSOC is designed by using task codes, direct acyclic graph and by the constraints of RFU. For every task this methodology determines which processor to select and corresponding configuration bits. Methodology for MRPSOC is described in four steps(fig 4.4):

1. Profiling step.
2. Identifying step.
3. Optimization step.
4. Assignment step.

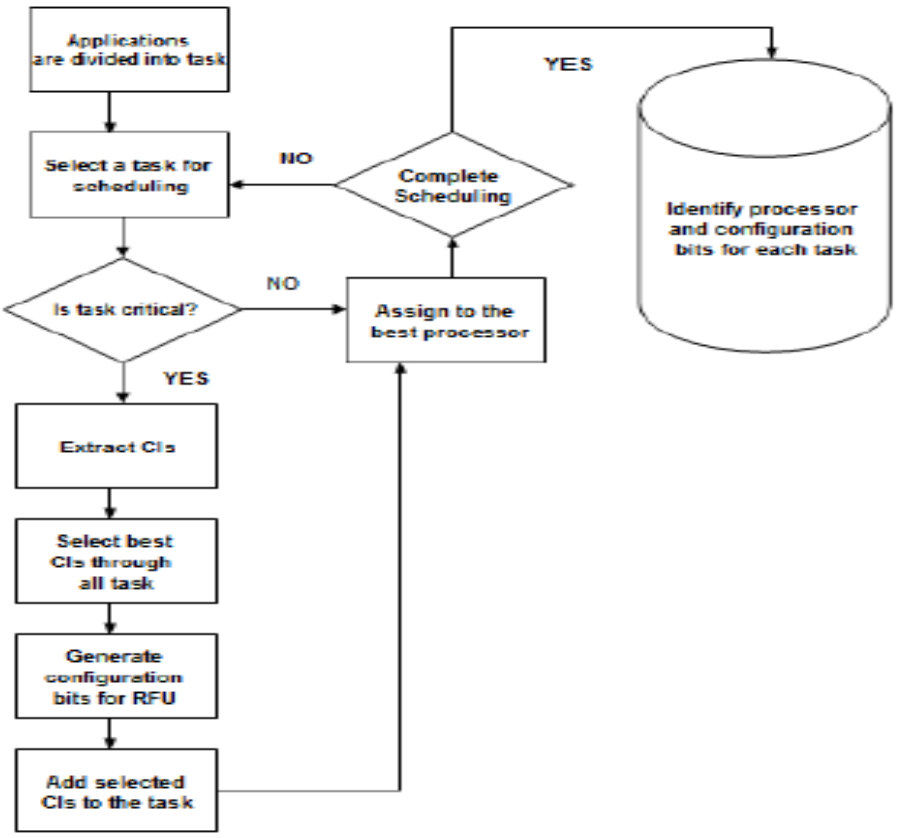

Fig 3: Methodology for MRPSOC [1]

1) Profiling step: This is the first step where the codes of the applications are run by direct acyclic graph on homogenous MPSOC without RFU. Identification of hot spots and execution time of particular application can be find out by profiling the tasks. Usage of any reachable solution can be done in profiling step since this step is independent of proposed methodology.

2) Identifying step: The task which is on critical path is identified in this step. Iterative improvement process is used in this step. Tasks are scheduled in each iteration. For scheduling tasks DCP algorithm is used. By using DCP algorithms concept, the task which are on critical path are identified. AEST (absolute earliest short time) and ALST (absolute latest short time) are the two concepts which are used by DCP algorithm. If AEST is equal to ALST then task is identified as critical one.

3) Optimization step: If custom instructions are selected for critical task then execution time can be reduced. But generating custom instructions itself is a complex procedure. This procedure includes searching patterns of the instruction set in a large space mainly to improve performance. So automatic tools are required to generate custom instructions or else it is difficult to generate CI's (custom instruction).

When CIs are identified, then this methodology generates configuration bits for reconfigurable function unit. Selections of CIs are done which must be accepted by RFU. But custom instruction parameters are much bigger when compared to RFU. Suppose ten operations are supported by RFU then CIs may go beyond this number. Based on the facilities which are available, the synthesis algorithm should be able to generate configuration bits by considering limitations.

The synthesis algorithm is executed in seven steps to generate configuration bits for all chosen CIs. At the beginning, mapping of CIs on the RFU is done by synthesis algorithm with no restriction. Then limitation of CIs is checked by considering restrictions. The size of CIs should not exceed RFU. If the limitation is crossed by CIs then synthesis algorithm should minimize operations in the particular level. Elimination of all ancestors/grand children is done in synthesis algorithm to avoid the problem. Elimination is based upon which one is less to remove weather ancestors or grand children. Then the number of levels , input and output numbers of executable CIs on the RFU are altered. Since different assignments in the algorithm are presented at different execution times, the attachment of configuration bits is done to task rather than processor. On the any available RISP the task is placed as a result of synthesis algorithm.

4) Assignment step: When the selection of CIs and generation of configuration bits are finished, assignment of task to the proper processor is done by DCP algorithm with new execution time. For the critical task, the algorithm assigns processor to task parent, if task is not critical then for any available task the appropriate processor is assigned.

Best CIs and the most critical tasks are selected by the algorithm in all possible iterations. But the task scheduling is affected by optimization routine custom instructions used by this will minimize completion time taken by the tasks.

The process is continued until platform exhibits no better performance or else no more CIs are added to the task because of the limitations present in memory configurations. In the succeeding iteration, more CIs are added to critical tasks. It is observed that the task which was critical in earlier iteration is not being critical in succeeding iterations.

D. ALGORITHM for MRPSOC

- First step is profiling step. Here the codes of the application are run by DAG (direct acyclic graph) on homogeneous MPSOC.
- Identification of hot spots and execution time of application is find out in profiling step.
- All the tasks are initialized after profiling step.
- Then initialized tasks are selected for scheduling.
- The task which is on critical path is identified in this step. Iterative improvement process is used in this step.
- If the task is on critical path then extract the custom instructions.
- After extraction of CIs, choose the best CIs through the entire task. The size of CIs should not exceed.
- Then generation of bit streams for configuring RFU's is done by considering RFU limitations.
- For the selected task execution time is modified, then add the selected CIs to the task.
- This task is assigned to the best processor with new execution time.
- Then scheduling of the task is completed for available CIs and memory, if not again identify the processor and configuration bits for each task.
- If the selected task is not on critical path then by using DCP algorithm the task is directly assigned to the best processor and scheduling of the task is completed.

E. Data level parallelism

Data level parallelism is one of the types of parallelism computing present in multiple processors. On various parallel computing nodes the data is distributed in case of data level parallelism.

Packaging of multiple data elements into single register which forms a vector can be done to exploit the data level parallelism. At a time the processor can run operation on the vector which is formed by data level parallelism. Multiple ALU's can also be provided to further exploit the parallelism. If scalar to vector conversation of data is done then efficiency of data level parallelism increases.

F. Instruction level parallelism

MRPSOC processor itself exploits instruction level parallelism. In consecutive instructions, the data dependence or control dependence is not present. Parallel execution of multiple instructions is possible. Using the superscalar technology exploitation of instruction level parallelism is done by the processor. Execution speed is increased by using instruction level parallelism since it allows to process more than two instructions at a time. Multiple numbers of memory requests are allowed by alleviating average memory latency in instruction level parallelism. If we use multigrain parallelism more than four instructions can be executed.

Concurrently all the tasks are executed at a time. For example (fig 4.5),

A= (n*n) + (m*m) + (o*o)

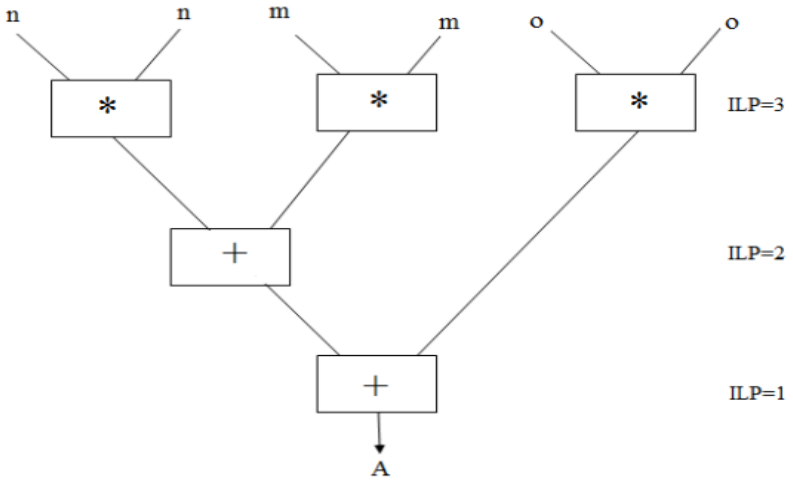

Fig 4: example for instruction level parallelism

G. Memory level parallelism

Pending operations can be done at a time by using memory level parallelism. Pending of operations usually occurs due to any cache missing or buffer missing. By the technique of pre-fetch, memory transfer level parallelism is achieved. Pre-fetching can be occurred in two ways it may be either initiated by hardware or initiated by software.

Software pre-fetching is used in this project. Identification of given application which needs a specific set of data is basic step of software pre-fetching. With the help of special instructions present in memory level parallelism, the pre-fetching of instruction is done to get the data in advance. By using pre-fetch less time is consumed for instruction execution to store the data.

III.SIMULATION RESULTS

In this paper the computation speed of enhanced MRPSOC processor is examined with the set of instructions. The implemented processor is compared with MPSOC to show that interfaced processor executes the instructions at faster rate than MPSOC.

1. Simulation of MPSOC

The MPSOC takes four instructions to reset the processes that are running before. To start the execution of any processor, this is the first step. After initialization the execution starts at pc=5 where first instruction is fetched at 33ns. As shown in fig 5(a).

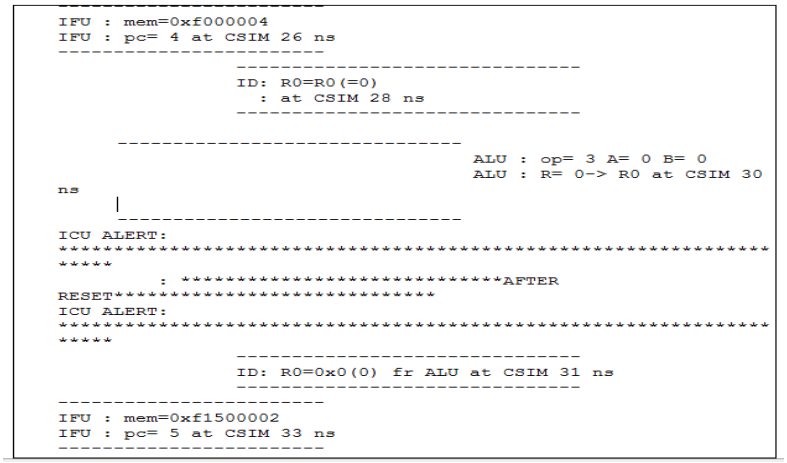

Fig 5(a): initializing the instructions and fetching the instruction (at pc=5) in MPSOC

In case of MPSOC only two operands can be given to the processor. For example consider instruction which is executing at 63ns. Here opcode is '3' so addition operation is done. Operands are 'A' and 'B' where the value of A=2 and B=0. The resultant is stored in register R3. As shown in fig 5(b).

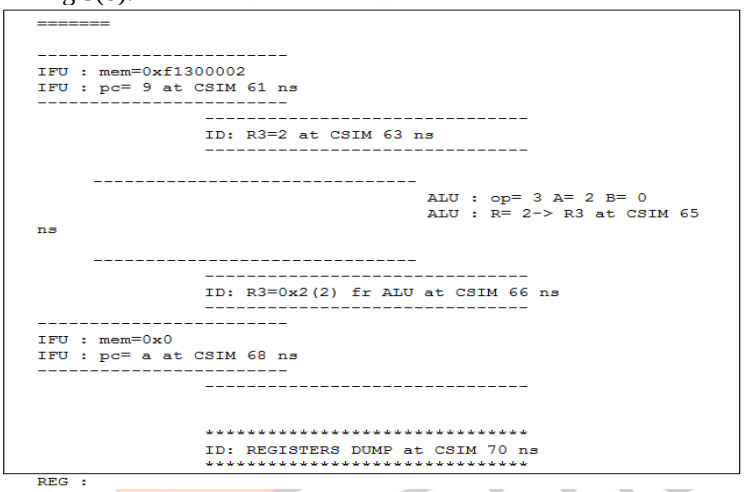

Fig 5(b): example of an instruction to show MPSOC can take two instructions at a time

The execution time taken by MPSOC is 173ns after executing all the instruction which is shown in figure 5(c). After completion of every instruction the result is stored registers which can also be seen in below figure.

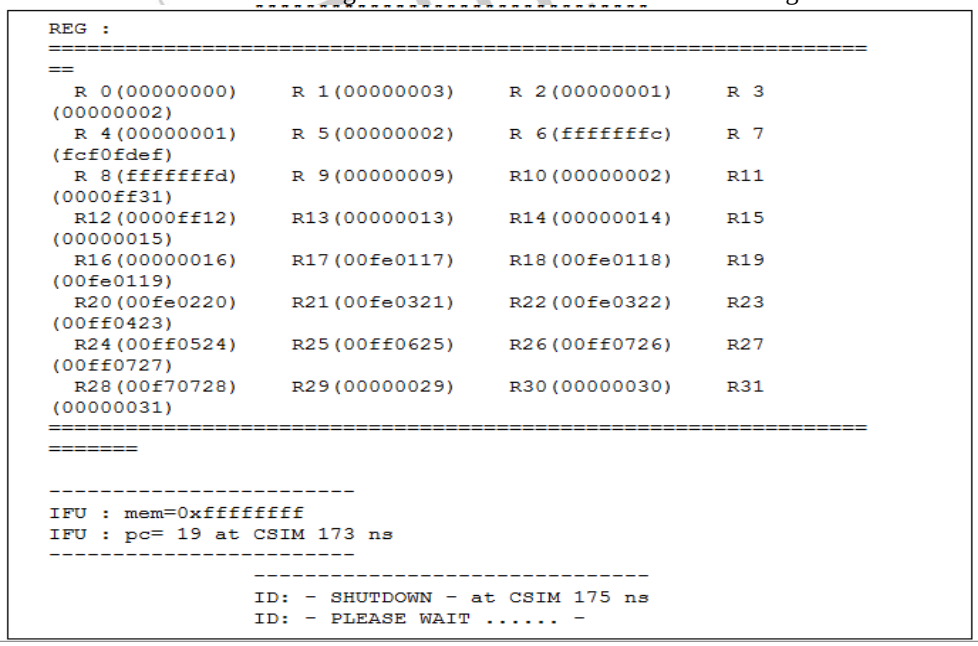

Fig 5(c) : execution completion time in MPSOC

2. Simulation of integrated processor (MRPSOC and Multigrain parallelism)

In case of implemented processor it is possible to give more than two operands. For example consider instruction which is executing at 51ns. Here opcode is '3' so addition operation is done. In this example operation is done on four operands. Operands are 'A', 'B','D' and 'E' where the value of A=1, B=0, D=1 and E=1. The resultant is stored in register R3. So we can conclude that by parallelism this processor is able to take more than four inputs (fig 5(d)).

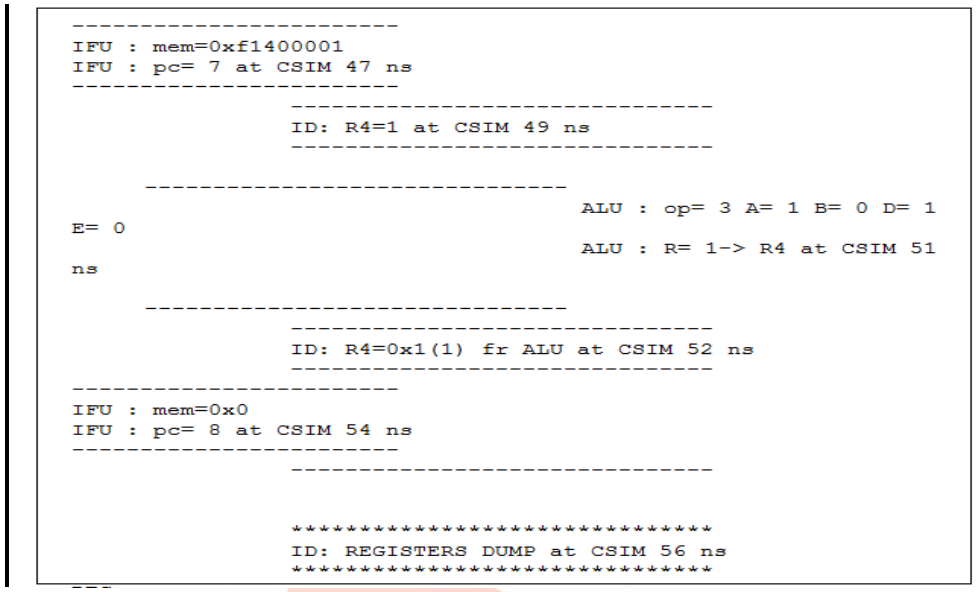

Fig 5(d): example of an instruction to show integrated processor can take more than two instructions at a time

The execution time taken by implemented processor is 126 ns after executing all the instruction which is shown in figure 5(e). After completion of every instruction the result is stored registers which can also be seen in fig 5(e). By the analysis of fig 5(d) and fig 5(e) we can conclude that time taken by implemented processor is less than MPSOC. Due the presence of MRPSOC the execution speed is more in case of implemented processor when compared to MPSOC.

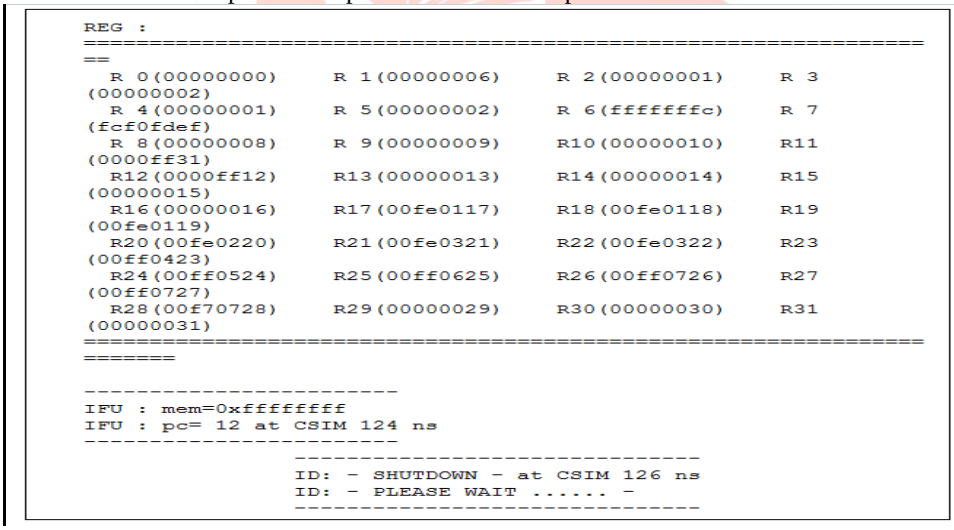

Fig 5(e): execution completion time in integrated processor

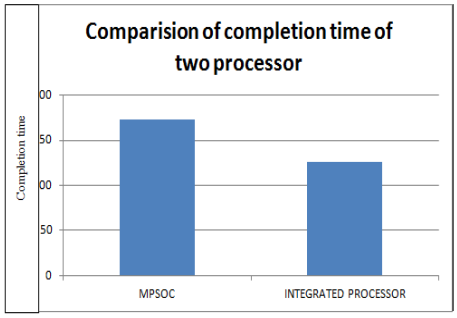

Fig 5(f): comparison of MPSOC and integrated processor with respect to completion time

The above graph (fig 5(f)) shows the comparison between implemented processor and MPSOC. To execute all the instructions MPSOC takes 175ns where as designed processor in the project takes 126 ns. So by analysis of the results, it is proven that integrated processor has faster speed than MPSOC.

IV.CONCLUSION

In this paper multigrain parallelism is integrated with MRPSOC. This adds extra feature to MRPSOC. Even though dispatch time of MRPSOC is less compared to MPSOC, still faster computation speed can be achieved by using parallelism. Results clearly show that compared to MRPSOC processor the designed processor is able to fetch more than two inputs simultaneously. Fetching rate is faster since at a time multiple data is fetched to the processor. Completion time of designed processor is very less compared to earlier processors. In system C the code is written and compilation is done in perl compiler. Cygwin is the software where the results of output can be seen.

ACKNOWLEDGMENT

We thank the referees for their suggestions to improve the content of this paper. The authors would thank the Principal for providing necessary facilities to carry out the work. We would thank our Head of the Department for guiding us towards the implementation.

REFERENCES

[1] R.Soleymanpour, Siamak Mohammadi, "A Platform for Multi Reconfigurable Instruction Set Processor System On Chip", in CSI International Symposium, 2013, pp.99-104

[2] Xiaoping Huang, Xiaoya Fan, Shengbing Zhang And Liwen Shi, "Investigation On Multi-Grain Parallelism In Chip Multiprocessor For Multimedia Application", proceeding of IEEE 2009, Computer School, Northwestern Polytechnical University,China.

[3] Ichiro Kuroda, Takao Nishitani, "Multimedia Processors", Proceedings of the IEEE,Vol.86,No.6, June 1998.

[4] F. Barat and R.Lauwereins, **"**Reconfigurable Instruction Set Processors: A Survey",Rapid system prototyping, 11th International workshop, 2000, pp.168- 173.

[5] L. Pozzi, K. Atasu, and P. Ienne, "Exact and approximate algorithms for the extension of embedded processor instruction sets," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.25, no. 7,pp. 1209-1229,2006.

[6] Xiaoping huang, xiaoya Fan,Shengbing Zhang, "The integration of multimedia process unit into an embedded processor", Proceeding of the 2007 IEEE international conference on integration technology,pp:492-495,March 2007,China.

[7] Jack L.Lo and SusanJ.Eggers; "Improving Balanced Scheduling with Compiler Optimization that Increase Instruction Level Parallelism", Department Of Computer Science and Engineering, University Of Washington, 1995.