An Efficient Application Specific Memory Storage and ASIP Behavior Optimization in Embedded System

Ravi Khatwal
Research scholar
Department of computer science
MLS University
Udaipur, India

Manoj Kumar Jain
Professor
Department of computer science
MLS University
Udaipur, India

***Abstract*—Low power embedded system requires effective memory design system which improves the system performance with the help of memory implementation techniques. Application specific data allocation design pattern implements the memory storage area and internal cell design techniques implements data transition speeds. Embedded cache design is implemented with simulator and scheduling approaches which can reduce the cache miss behavior and improve the cache hit quantities. Cache hit optimization, delay reduction and latency prediction techniques are effective for ASIP design. The design functionality is simply specifying the tradeoff among various design metrics like performance, power, size, cost and flexibility. ASIP behavior and memory storage area optimized for low power embedded system and implements cycle time with effective scheduling techniques which implements the system performance with low power consumption.**

Keywords*—Memory design; Compiler; Processor design; Scheduling Techniques; Memory storage

I. INTRODUCTION

Embedded systems uses some specific constraints such as Real time design metrics are a measurement of application features such as Cost, Size, Power and High Performances. Reactive and real time required to implement our system environments and computed application results in real time without any delay [Fig. 1]. Currently embedded system designer are being designed on a silicon chip and also design for critical applications like killer application (smart phone), smart card, video game, mobile internet, handheld embedded system, GBPS device, gigabyte per second LAN system. Embedded design technologies used to improve the design technology to enhance productivity has been a focus on software and hardware design mechanism.

In HLS design mechanism, Xilinx simulator software is used to verify all the functionality and timing custom peripheral design architecture [18, 20]. ASIP design used to implement the functional unit may then either be integrated on a chip or implements peripheral devices. Profiler is effectively used in Pre-allocation memory design and implements pre-allocation based execution delay time. Recently a memory implementation technique is attracting strong research interest in ASIP. ASIP is a heterogeneous platform composed of programmable processor core and used customized hardware environments [1, 2, 3]. ASIC architecture is not flexible for specific application design architecture. DSP processor is also flexible and fully programmable; it can't achieve high performance with low power consumption and not suitable for various complex application development mechanisms.

VLIW processor unit require compiler support and VLIW architecture is characterized by instructions such that each specifies several independent operations. This is compared to RISC instructions that typically specify one operation and CISC instructions that typically specify the several operations with sufficient registers, A VLIW machine can place the results of speculative executed instructions in temporary registers. The level of sophistication in VLIW compiler is significantly higher.

The heterogeneous vector width method use to expose the heterogeneous vector widths for VLIW ASIP [10, 13]. Effective automation is analyzed for VLIW ASIPs. The lower bound latency is effective for VLIW ASIP. Latency bound mechanism implements the data transfer delays [9]. By the help of these approaches a window data flow graph and lower bound deign mechanism reduce the delay penalties due to operation serialization or data transfer mechanism.

An effective emulation tool chain designed for ASIP design architecture [5]. The FPGA based emulator is alternative to pure software cycle-accurate simulation and this tool chain to reduce the design exploration time [13]. Fast and accurate processor simulator used for high performance ASIP simulation [4] and an integrated tool chain design also evaluated for ASIP systems [5]. ASIP architecture also design for a Discrete Fourier transform (DFT)/Discrete cosine transform (DCT) /Finite impulse response filters (FIR) engine[14].

Memory data storage and operational optimal delay frequency analyzed according application computational conditions. Embedded process system analysis is presented in the next section. Section 3 and 4 represents the application specific data storage and data storage is effectively optimized in memory system. Last section represents application specific data storage in ASIP system and implements system performances with various techniques such as delay reduction, latency prediction and operational scheduling mechanism.

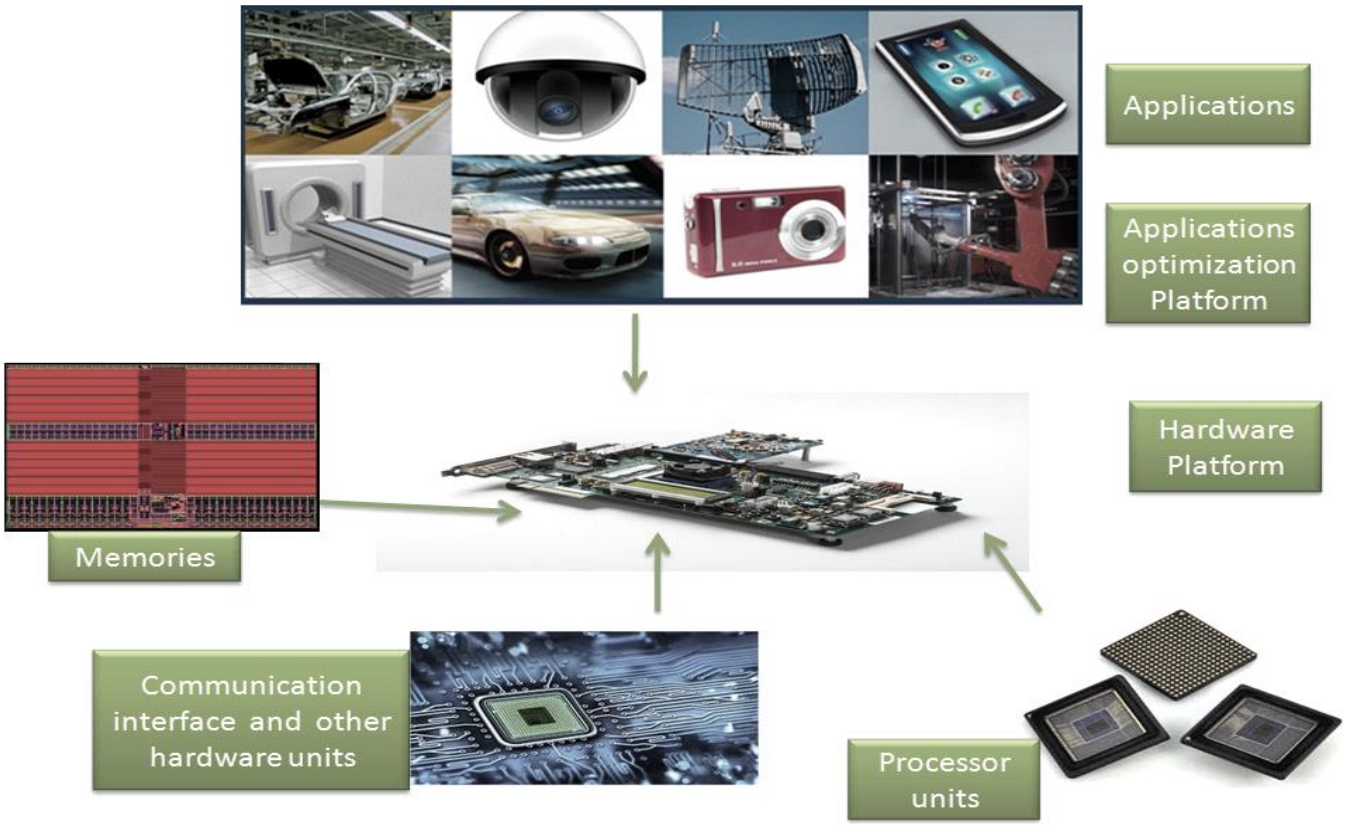

Fig. 1. Application Specific Requirements Based Embedded System Design

II. Application Specific Embedded Process Analysis

The basic process of embedded system is implemented with three basic mechanisms such as application compilation, synthesis and implementation, IP based integration and test and verification by specific simulator. By the help of this mechanism we implement the application based embedded systems design for low power embedded devices. In embedded system the HLS design mechanism Memory designer used high level language and implements behavioral specifications into register-transfer (RT) specifications by converting behavior on general-purpose processors to assembly code. The memory Designer also refines the register-transfer-level specification of a single-purpose processor into a logic specification and finally implements machine code for general-purpose processors and utilizes the gate-level net list. First Compilation/Synthesis process the designer specifies desired functionality in an abstract manner. A compiler translates the source language into its target machine language without having the option for generating intermediate code. Each new machine have a full native compiler is required [Fig. 2]. The Software compiler converts a sequential program to an assembly code, which is essentially a register-transfer code and a system synthesis tool converts an abstract system specification into a set of sequential programs on general and single-purpose Processors. A logic synthesis tool converts Boolean expressions into a connection of logic gates (called a net list). A register-transfer (RT) level synthesis tool converts finite-state machines and register-transfers into a data path of RT components and a controller of Boolean equations. A behavioral synthesis tool converts a sequential program into finite-state machines and register transfers.

Second **Libraries/IP based implementation** phase is used the logic-level library and it consists of layouts for gates and cells. The RT-level library may consist of layouts for RTL components, like registers, multiplexers, decoders, and functional units. A behavioral-level library may consist of embedded components, such as compression components, bus interfaces, display controllers, and even general- purpose processors. IP integration design is used to implement the memory or various peripheral devices and integrating the device according to our application requirements. Finally, a system-level library might consist of complete systems, solving particular problems, such as an interconnection of processors, memory with accompanying operating systems and programs to implement an interface.

Finally, **Test/Verification** phase we have analyzed the functionality of the design is correct or not and checked the mechanism with low abstraction levels to high abstraction levels. Simulation mechanism better utilizes the testing for correct functionality. The Logic level, gate- level simulators provides output signal timing waveforms with a given input signal waveform. And finally RTL level, hardware description language (HDL) simulators used to execute the RTL-level descriptions and provide output according to the given input waveforms. The behavioral level, HDL simulators used to simulate sequential programs and co-simulators connect HDL and processor simulators to enable hardware/software co-verification at the system level. Model simulator simulates the

initial system specification using an abstract computation model, that independently of any kind of processor technology and these simulators verify the correctness and completeness of the specification [Fig. 3].

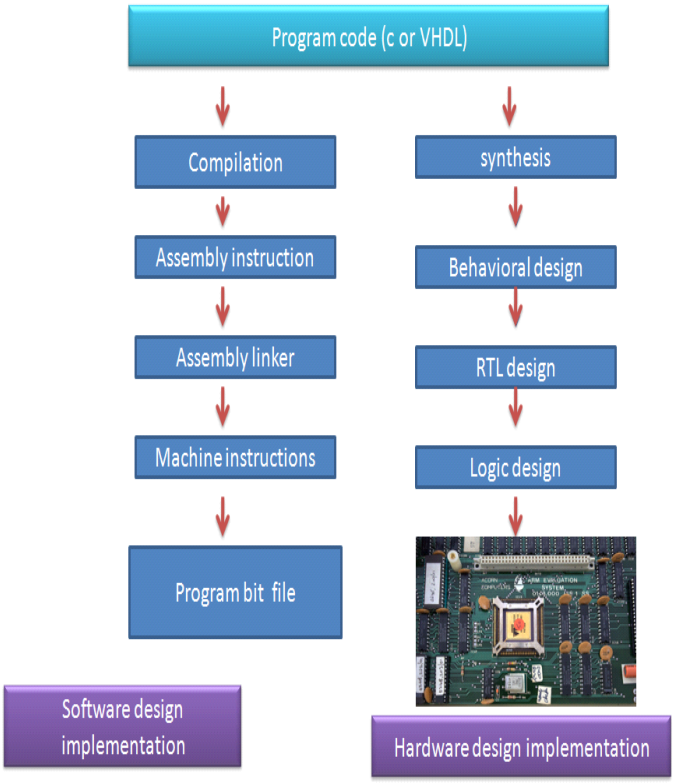

Fig. 2. Design process used in embedded system

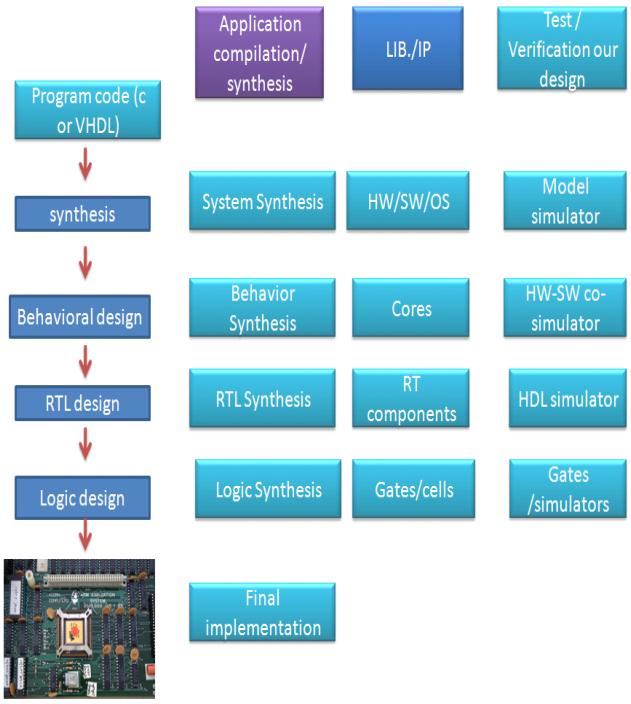

Fig. 3. Design process technology of embedded system

III. Appplication Specific Data Storage Analysis

Application specific computation frequencies analyzed according application behavior and their design complexity. Various standards application benchmarks used and analyzed the computation complexity with various critical conditions such as higher multiplication or lower multiplication. At First condition contains computation time O (n) loop used d n time's units and repeats a programming statement in n times. Second condition contains every iteration of the loop counter will be divided by 2 so computation designs as 2^4=16 words used. Third condition contains the nested loop used so computation designs complexity as O (n^2) and it represents a loop executing inside loops. Fourth condition contains operational computation complexity is O (n) loop independently of each other. Fifth condition contains computation complexity is O (n log n). Six condition computation complexity is 2^N complexity used due to loop multiply 2 so 2^N-1 possibility available and final condition have computation possibility in two part O (n) and O (n^2) available for memory allocation area. Critical benchmark application and their computation design complexity [Fig.4, Fig. 5, Fig. 6, Fig. 7 and Fig. 8]. Various critical application such as vision application, robotic memory design, mind mapping artificial neural network based application designs are implemented according to their computational complexity [Fig. 9 and Fig. 10].

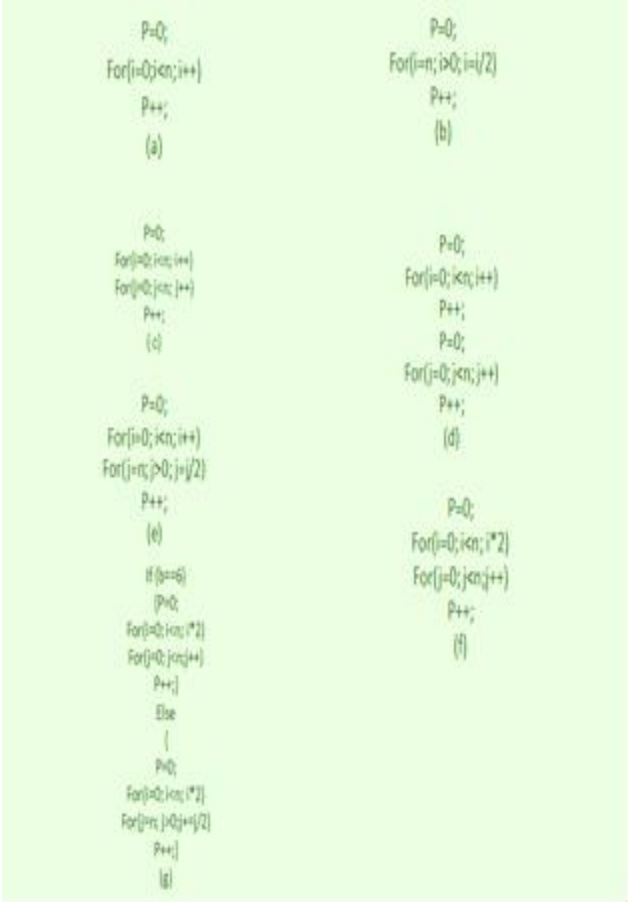

Fig. 4. Application specific operations frequency analysis with design complexity

```
*/
{
  int i;

  for ( i = 0; i < n; i++ )
  {
    x[i] = ( ( a + b ) + ( b - a ) * x[i] ) / 2.0;
  }
  for ( i = 0; i < n; i++ )
  {
    w[i] = ( b - a ) * w[i] / 2.0;
  }
```

Fig. 5. Patterson application design sections

```
cout << "CELLULAR_AUTOMATON:\n";
cout << "  C++ version.\n";

n = 80;
step_num = 80;

x = new char[n+2];
x_old = new char[n+2];

for ( i = 0; i <= n + 1; i++ )
{
  x[i] = ' ';
}
x[40] = '*';

for ( i = 1; i <= n; i++ )
{
  cout << x[i];
}
cout << "\n";

for ( j = 1; j <= step_num; j++ )
{
  for ( i = 0; i < n + 2; i++ )
  {
    x_old[i] = x[i];
  }
  for ( i = 1; i <= n; i++ )
```

Fig. 6. Cellular automation application section

```
for e = 1 : 17

  c = ( ( ( x - x0(e) ) * v11(e) ...
        + ( y - y0(e) ) * v12(e) ...
        + ( z - z0(e) ) * v13(e) ) / a1(e) ) .^2 ...
    + ( ( ( x - x0(e) ) * v21(e) ...
        + ( y - y0(e) ) * v22(e) ...
        + ( z - z0(e) ) * v23(e) ) / a2(e) ) .^2 ...
    + ( ( ( x - x0(e) ) * v31(e) ...
        + ( y - y0(e) ) * v32(e) ...
        + ( z - z0(e) ) * v33(e) ) / a3(e) ) .^2;

  i = find ( c <= 1.0 );

  f(i) = f(i) + g(e);

end

return
end
```

Fig. 7. 3D Shepp-Logan application design section

```
w = 1.0;

dc = ( double * ) malloc ( n * sizeof ( double ) );

for ( j = 0; j < n; j++ )
{
  dc[j] = 0.0;
}

while ( k < m )
{
  if ( k < m )
  {
    k = k + 1;
    for ( j = 0; j < n; j++ )
    {
      dc[j] = dc[j] + omega[k-1] * sin ( w * pi * x[j] );
    }
  }

  if ( k < m )
  {
    k = k + 1;
    for ( j = 0; j < n; j++ )
    {
      dc[j] = dc[j] + omega[k-1] * cos ( w * pi * x[j] );
    }
  }

  w = w + 1.0;

}
```

Fig. 8. Cycle reduction application design section

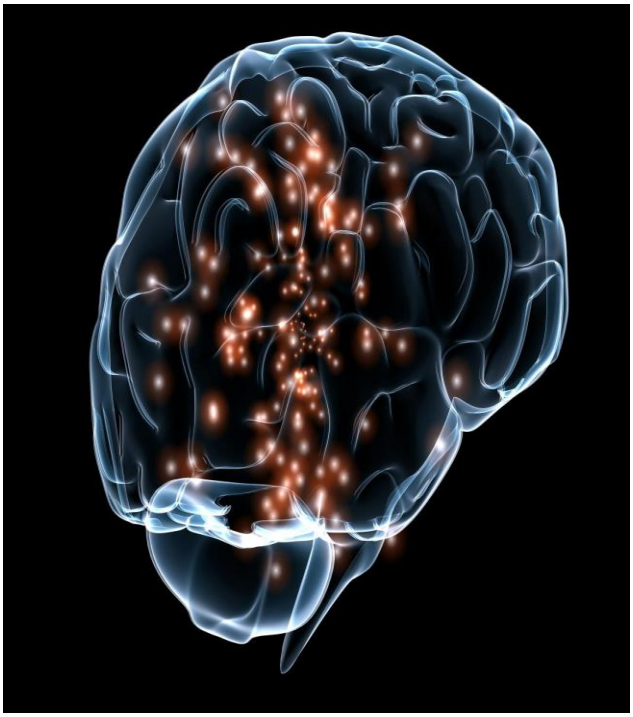

Fig. 9. Basic Mind mapping application section area

Fig. 10. Basic Embedded vision application section

IV. Data Storage Optimization in Memory System

Memory blocks are important concepts from both code generation and optimization point of view. Basic blocks play an important role in identifying variables (data), which are being used more than once in a single basic block. Memory blocks analysis with various schemes such as data flow, data allocation, data reusability, scheduling approaches and control flow design mechanism. When the first instruction is executed, then all the instructions in the same basic block will be executed in their sequence of appearance without losing the flow control of the program. The Blocking is another way of reordering iteration in a loop and greatly improves the locality of source code [Fig. 11 and Fig. 12]. Each memory element is implemented with the memory array design, architecture so block architecture design easily identify the data arrangements. Data reusability design is implemented by a References data storage mechanism. Matrix blocks are divided into sub blocks or sub matrixes and column implemented design effective reusable code design mechanism implements a cache hit condition [Fig. 16] [19].

Data stored in memory with the Serial execution mechanism of data elements depends upon row and column design architecture. Memory design is implemented with matrix level blocks architecture. In Matrix designs innermost loop reads and writes the same elements of z and use the row of x and column of y. Each block is designed according to the row & column accessing scheme [Fig. 12]. The Application based storage element is arranged in our block area and a single row is spread among only n/E element cache lines. When all data element is filling in the cache only n/E cache misses occur for a fixed value index and the entire total operation use n^2/E. If the cache is big enough that all n^2/E cache lines holding column Y can reside together in the cache, then no more cache misses [Fig. 14] occurred. Column index implementation technique implements the repositioning of memory data arrangements which reduces the cache misses or data cluster and it's easily serialize operational frequency. The total number of misses is depending upon $2n^2/E$, half for x and half of y. The Single processor will be computed n^2/E elements of Z; performing n^P/E where operation complexity p is changed according to application computation.

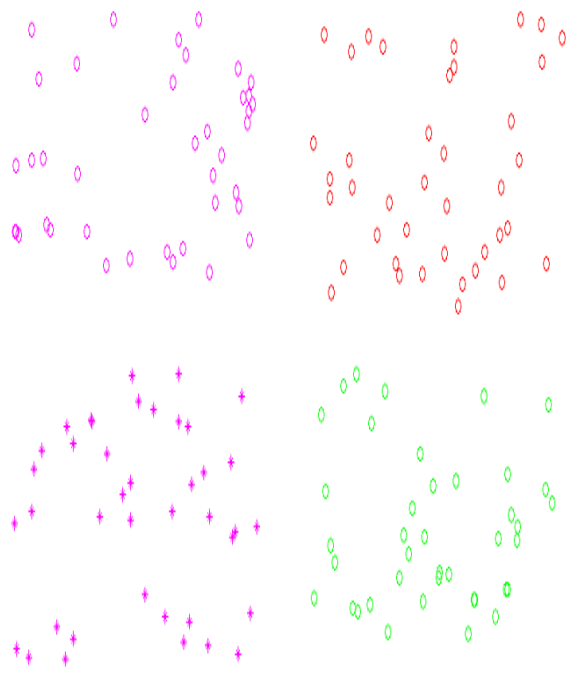

Fig. 11. Complex Cluster block analysis

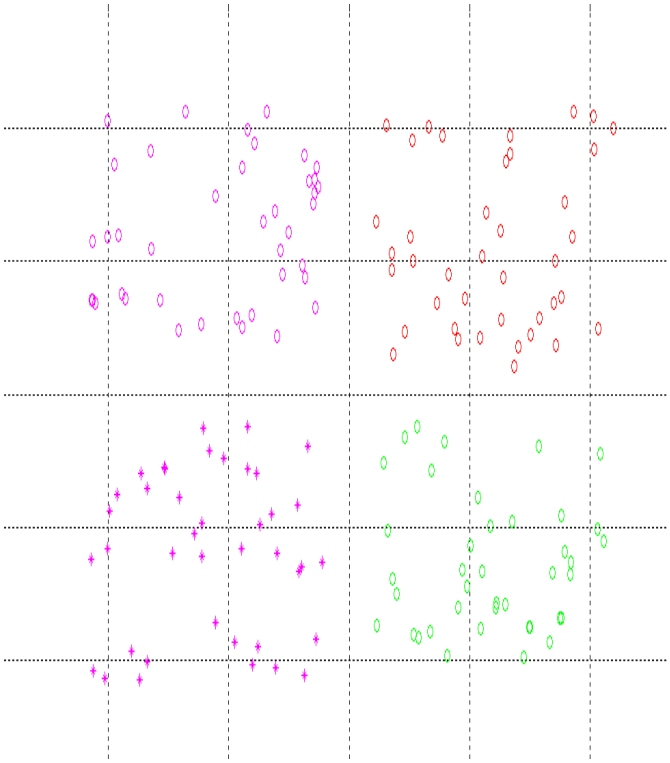

Fig. 12. Block filter mechanism which Increase data probability

A. Delay reduction design optimization

The critical path would be combination logic delay plus the logic circuits setup time, plus the clock output delay. The critical path analysis with various nodes based implementation. Complex memory structures have various critical sections. Various critical paths cell delay analyzed and combinational path delay is implemented with column based cell architecture. Application based column design implements cycle reduction and this design is used for data allocations which can implement data shifting and reduce the memory misses [Fig.16]. We have analyzed the performance based lower and higher frequency order based access time variations used in memory implementation mechanism [Fig. 13, Fig. 14 and Fig. 15]. The probability degree based access time pattern implements the critical section area. Higher critical section area has longer access time probability and it takes longer access time [Fig. 16]. Various approaches such as scheduling, allocation and binding pattern implements the access time and have a low probability frequency design which reduces the critical section area [Fig 17 and Fig. 18]. Node based critical section is implemented the high and lower order path for access time point of view and column implemented shortest path have lower access time path which have global impact in system performances.

```
    cout << "  Zero diagonal entry, index = " << i << "\n";
    exit ( 1 );
  }
}

if ( job == 0 )
{
  for ( it_num = 1; it_num <= it_max; it_num++ )
  {
    x[0] =   ( b[0]                  - a[2+0*3] * x[1]     ) / a[1+0*3];
    for ( i = 1; i < n-1; i++ )
    {
      x[i] = ( b[i] - a[0+i*3] * x[i-1] - a[2+i*3] * x[i+1] ) / a[1+i*3];
    }
    x[n-1] =   ( b[n-1] - a[0+(n-1)*3] * x[n-2]             ) / a[1+(n-1)*3];
  }
}
else
{
  for ( it_num = 1; it_num <= it_max; it_num++ )
  {
    x[0] =   ( b[0]                    - a[0+1*3] * x[1]     )
         / a[1+0*3];
    for ( i = 1; i < n-1; i++ )
    {
      x[i] = ( b[i] - a[2+(i-1)*3] * x[i-1] - a[0+(i+1)*3] * x[i+1] )
           / a[1+i*3];
    }
    x[n-1] =   ( b[n-1] - a[2+(n-2)*3] * x[n-2]                    )
           / a[1+(n-1)*3];
  }
}
```

Fig. 13. Diagonal matrix application design sections

```
k = seed / 127773;

seed = 16807 * ( seed - k * 127773 ) - k * 2836;

if ( seed < 0 )
{
  seed = seed + i4_huge;
}

r = ( float ) ( seed ) * 4.656612875E-10;
/
/  Scale R to lie between A-0.5 and B+0.5.
/
r = ( 1.0 - r ) * ( ( float ) a - 0.5 )
  +         r   * ( ( float ) b + 0.5 );
/
/  Use rounding to convert R to an integer between A and B.
/
value = round ( r );
/
/  Guarantee A <= VALUE <= B.
/
if ( value < a )
{
  value = a;
```

Fig. 14. Data allocation application design section

```
while ( 1 < il )
{
  ipnt = ipntp;
  ipntp = ipntp + il;
  il = il / 2;
  ndiv = ndiv * 2;

  for ( j = 0; j < nb; j++ )
  {
    ihaf = ipntp;
    for ( iful = ipnt + 2; iful <= ipntp; iful = iful + 2 )
    {
      ihaf = ihaf + 1;
      rhs[ihaf+j*(2*n+1)] = rhs[iful+j*(2*n+1)]
        - a_cr[2+(iful-1)*3] * rhs[iful-1+j*(2*n+1)]
        - a_cr[0+iful*3]     * rhs[iful+1+j*(2*n+1)];
    }
  }
}

for ( j = 0; j < nb; j++ )
{
  rhs[ihaf+j*(2*n+1)] = rhs[ihaf+j*(2*n+1)] * a_cr[1+ihaf*3];
}

ipnt = ipntp;

while ( 0 < ipnt )
{
  ipntp = ipnt;
  ndiv = ndiv / 2;
  il = n / ndiv;
  ipnt = ipnt - il;
```

Fig. 15. Cycle reduction application design section

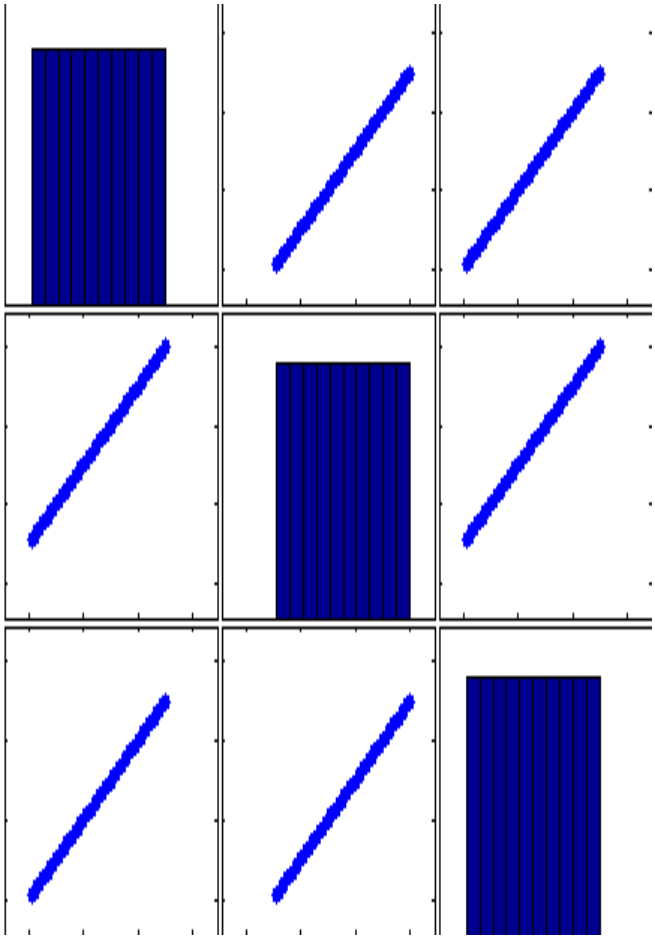

Fig. 16. Data filtering according to Diagonal design mechanism

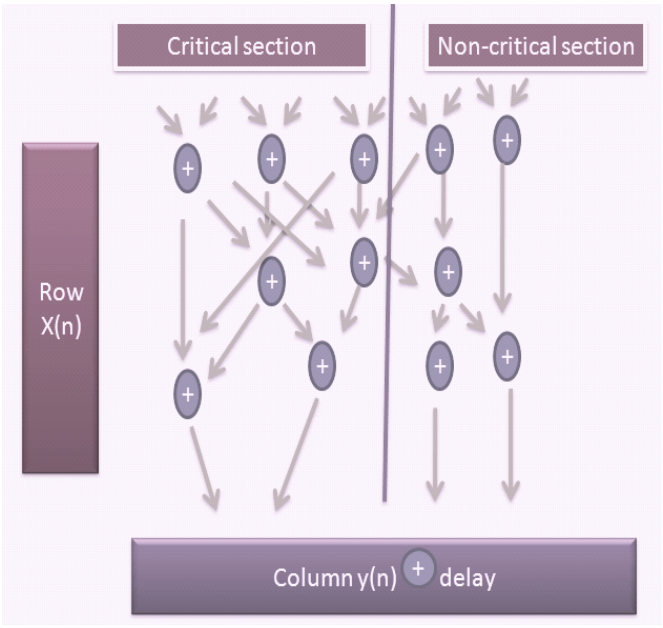

Fig. 17. Critical or non critical delay design section

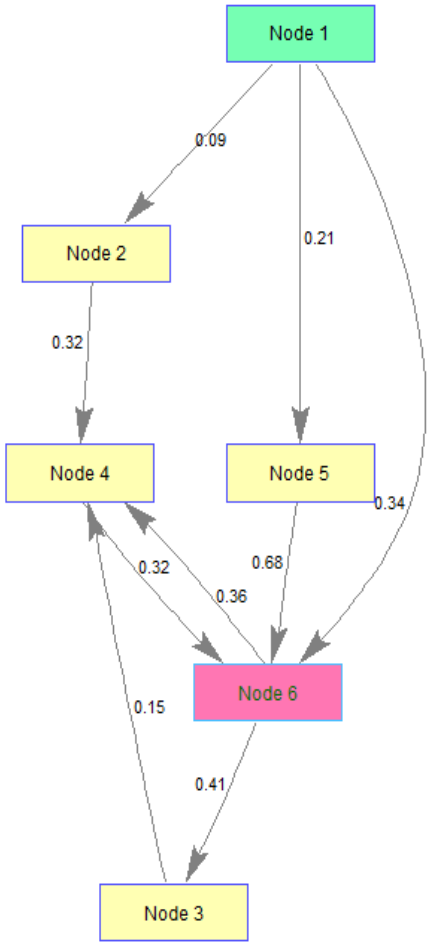

Fig. 18. Lower and higher delay point based critical section analyis

B. Scheduling approaches for reduction of memory space

Scheduling techniques are required to schedule the memory operations and operation scheduling effective determine the memory cost area. The scheduling algorithm will attempt to parallelize the operation to meet the timing constraints and scheduler mechanism will serialize the operation to meet the resource constraints [17]. Various scheduling problem implemented with different requirement such as time constraints, resource constraint, feasibly constrained [19, 20]. Memory operation scheduling implemented with three conditions such as FCFOP (First Come First Operational), LCLOP (Last Come Last Operational) and operational optimal degree based operational [Fig. 19, Fig. 20 and Fig. 21]. Scheduling approaches implement according to some conditions time, resource and feasible levels. The max no. of time step finds the cheapest schedule which satisfied the constraints. Lower resources find the fastest with satisfied the constraints [Fig. 22]. Feasible conditions decide if there exists a schedule which satisfied the constraints or not.

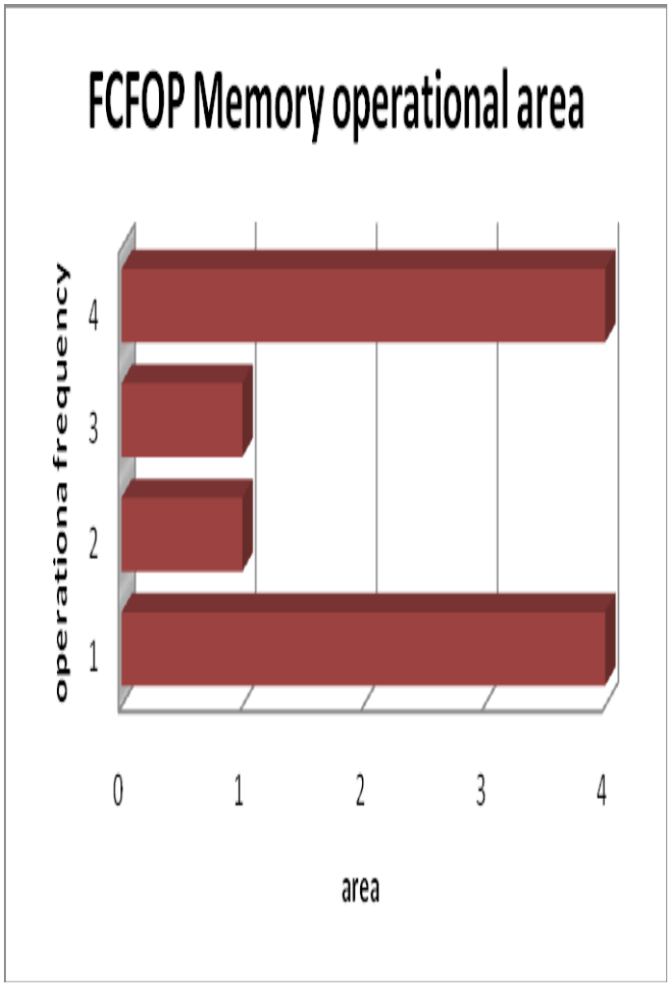

Fig. 19. FCFOP scheduling

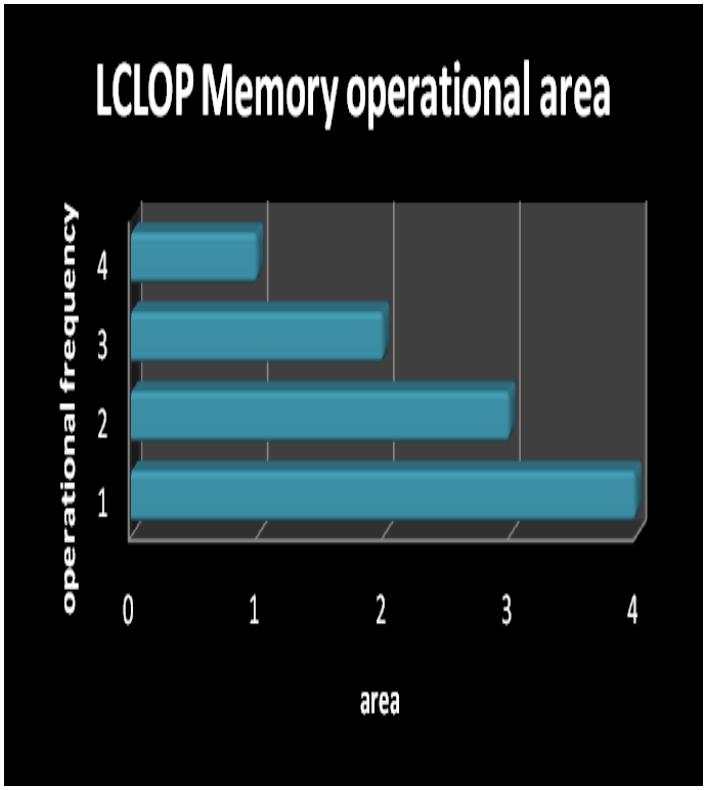

Fig. 20. LCLOP scheduling

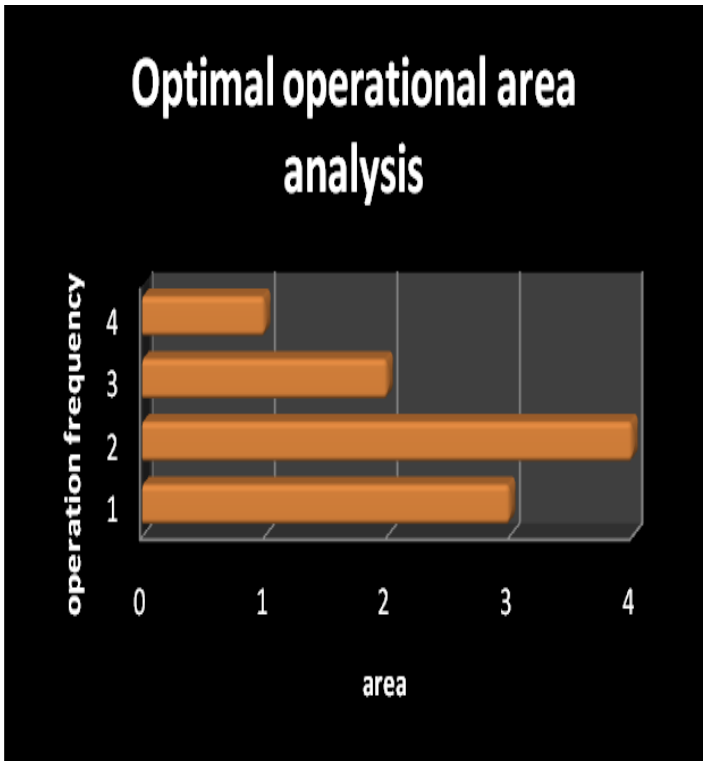

Fig. 21. Optimal design complexity based operational Scheduling

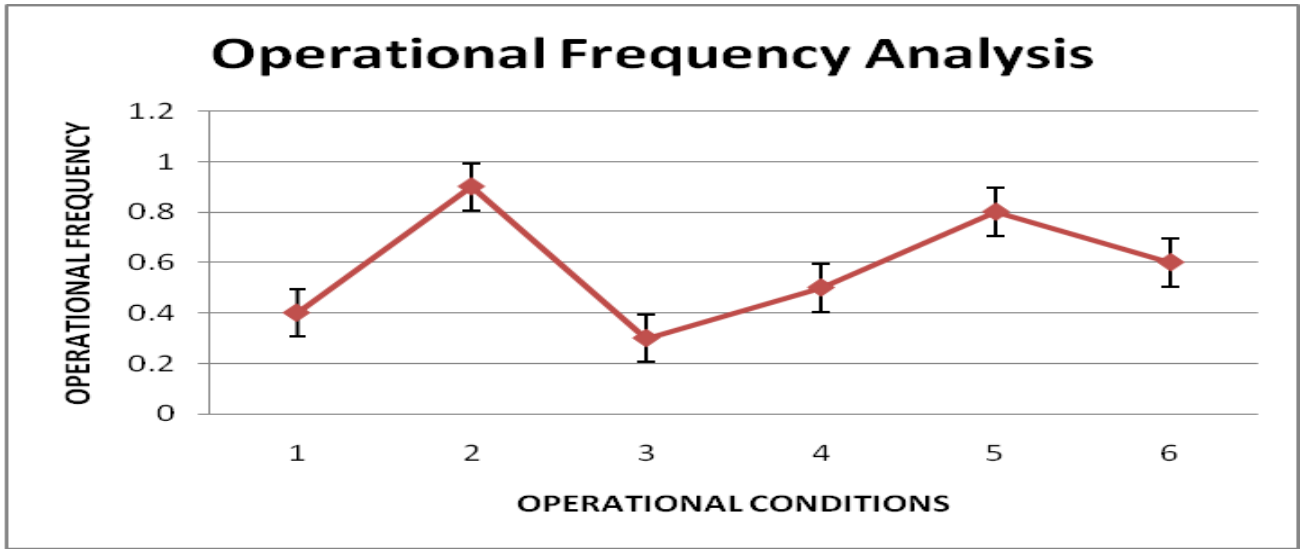

Fig. 22. Operational optimal frequency analysis

V. ASIP Behavior Optimization in Embedded System

Memory optimization techniques and performance area is determined by standards benchmark application. An application specific memory simulation analyzed by various simulators such as trace driven, cheetah, cache, ARM DS-5 etc. The advantages of SRAM used in programming technology so designer reuse the chip during prototyping and a system can be manufactured using in system programming.

In co-design technology effective memory performance area is analyzed by various simulators. The Co-design technology of ASIP used hardware and software implementation designs system to achieve an effective performance in the form of cycle count, low power consumption, latency and execution time [Fig. 23]. The source code profiling approach easily understands the application to guide the ASIP design methodology.

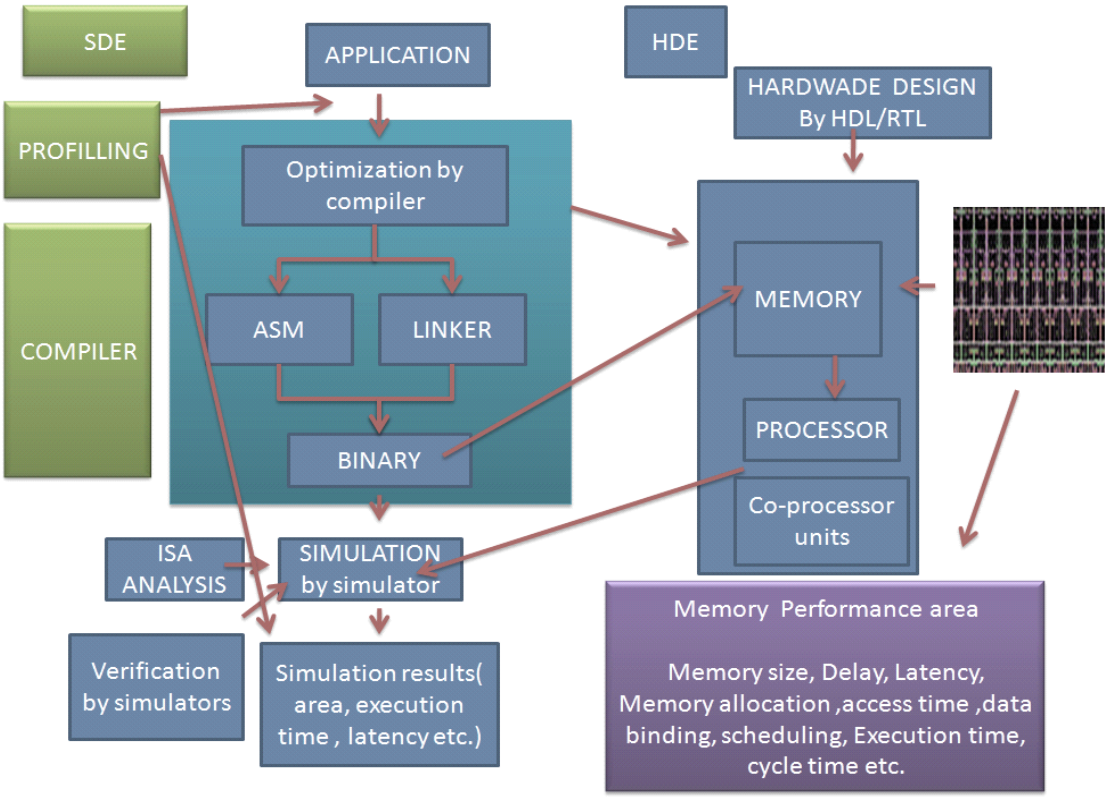

Fig. 23. Application specific Memory Integration and Performance Area

A. Application specific profiling and compilation Overview

Profiler have used to analyze the target source programs by collecting information on their execution based due to their data granularity scheme [10]. Profiler implements Pre-allocation of memory architecture and implements the execution time of application. A memory profiler used which implements dynamic profiling techniques to generate memory traces [10]. Memory object is computed load/store information for ASIP design mechanism. Micro-profiling approach also fills the gap between source level and instruction level profiler and implements speed and accuracy for ASIP design system [11]. LANCE [15] is mainly intended to facilitate C compiler design for embedded processors, so as to eliminate the need for time- consuming assembly programming [Fig. 24].Figure 24 shows the basic framework of LANCE profiler overview which requires profiling library, source code and instrumented binary file for profiling. Embedded processors for which LANCE based C compilers have been successfully built include both RISCs and DSPs design. The implementation of edge profiling, path profiling methods combines profiles with in the Low Level Virtual Machine [16] compiler infrastructure [Fig. 25]. A Codelets EXTRACTOR and RE player implements the code isolation. Codelet is basically designed for implementing, compiled, run and measure independently for the original application. The ISA design require an effectively for a fine grained profiling mechanism is based on C compiler mechanism.

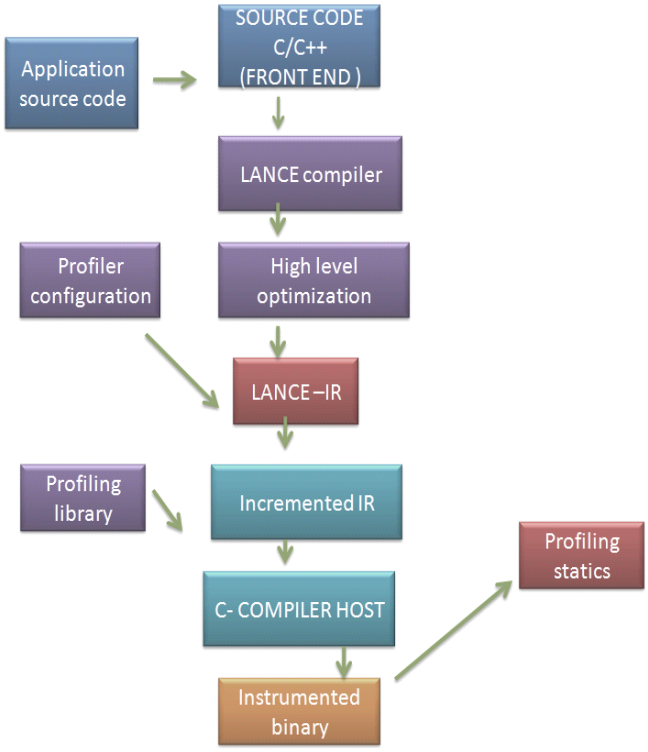

Fig. 24. LANCE Profiler in ASIP Embedded system [16]

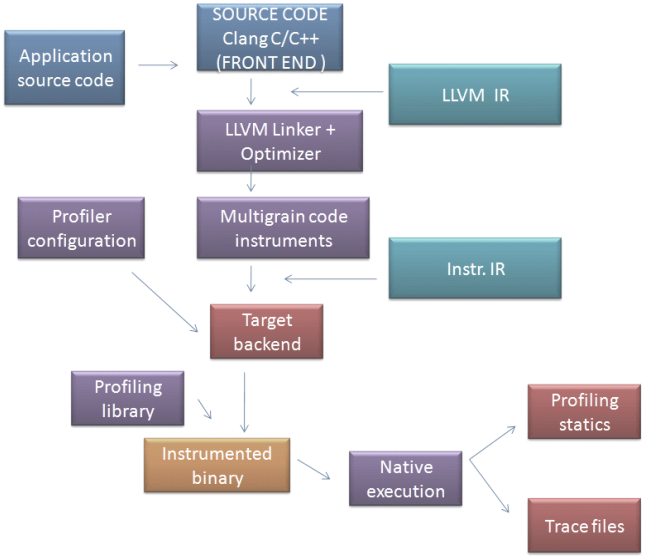

Fig. 25. LLVM based profiling analysis [17]

B. Application specific Latency prediction

Recently high level synthesis design is used efficient latency prediction techniques which implements the applications specific system performances and latency prediction design also used in clock cycle reduction mechanism or operational serialization. The number of time unit's clock cycles between initiations of stage is the latency between them. A latency of k means that the initiation are separated by k clock cycles. Any attempt two or more initiations to use the same stages at the same time they will cause a collision and collision must be avoided by scheduling a sequence initiations stages. In state diagram mechanism we have analyzed the function x from the initial stage (101010101110), only five outgoing transition are possible, corresponding to the five permissible latencies 10,8,6,4 and 1 in the initial collision vector. Similarly Free State (10101011), one reaches the same state offer three, five or seven shifts.

When condition is n+1 or greater, all the data transitions are redirected back to the initial states. A Collision can be implemented them by greedy cycles. Greedy cycles from the state diagram we can determine optimal latency cycles which result in the MAL. There are infinitely many latencies cycles, one can from state diagram, suppose that (1,12),(1,4,6,8,10,12),(4,6),(4,6,8)…… are legitimate cycles traced from the state diagram. As simple cycles are latency cycles in which each state appear only ones. Only (4),(6),(8),(6,8),(10,12) are simple cycles the cycles (6,12,10,12) are a complex cycle because of its travels these the states (101010101110) twice or more. Similarly (4,6,4,6,8,6) is not simple it repeats the state so we need greedy cycles is one whose edge are all made with minimum latencies from their respective starting states. The greedy cycles (1, 12) average latency is 6.5, which is lower than that of the simple cycle (10, 12) is 11[Fig. 26].

Greedy cycles have a constant latency which is equal the MAL (minimal average latencies points) for evaluating function X without causing collision the collision free scheduling approaches is thus reduced to finding greedy cycles from the sets of simple cycles. The greedy cycles yielding the MAL are the suitable choice for performance improvements. A latency sequence is a sequence of permissible forbidden latencies between the successive task initiations. A latency cycle is a latency sequence which repeats the same sequence indefinitely. Repeating of the cycles that reduces the collision between them and used the average latency that reduces the collision. Constant cycles contain is the latency cycles which contain only are latency value. The average latency cycles of a constant cycle are simple the latency itself. The target machine [RISC, CISC, and VLIW] can deploy more sophisticated instructions, which can have the capability to perform specific operations much efficiently.

If the target code can accommodate those instructions directly, that will not only improve the quality of code, but also yield more efficient results [Fig. 27]. Fixed point based latency optimal frequency optimized according various mechanism such as delay point and optimal operational frequency prediction mechanism. Operational serialize means how application computation complete the task with the least waste of time or least waste of hardware resources. Optimal condition is required to serialize the computational operations so resource reducible or operational optimal condition implements the latency design for ASIP system.

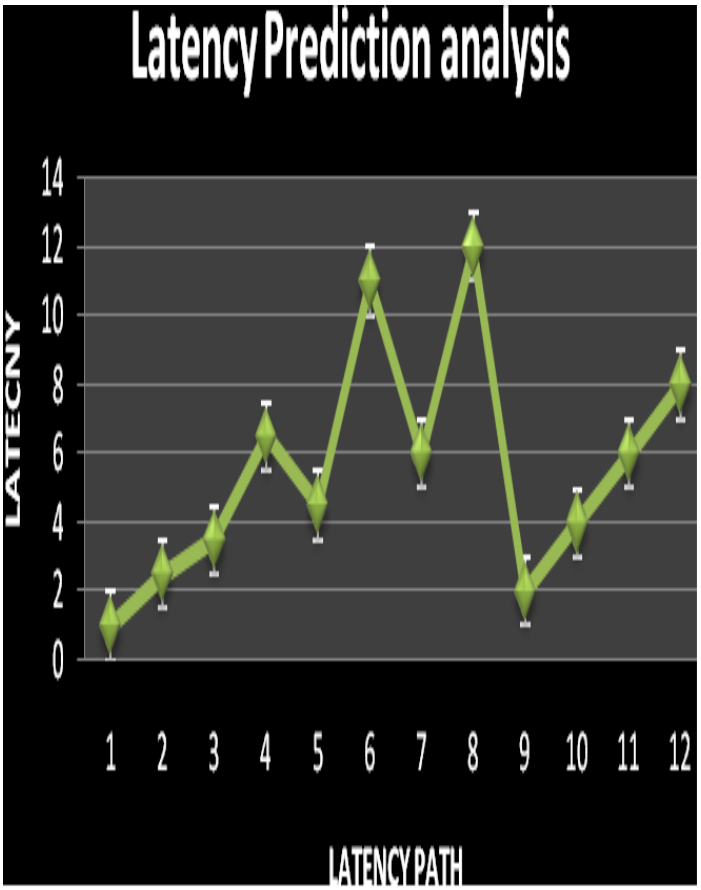

Fig. 26. Greedy cycle based LATENCY prediction analysis

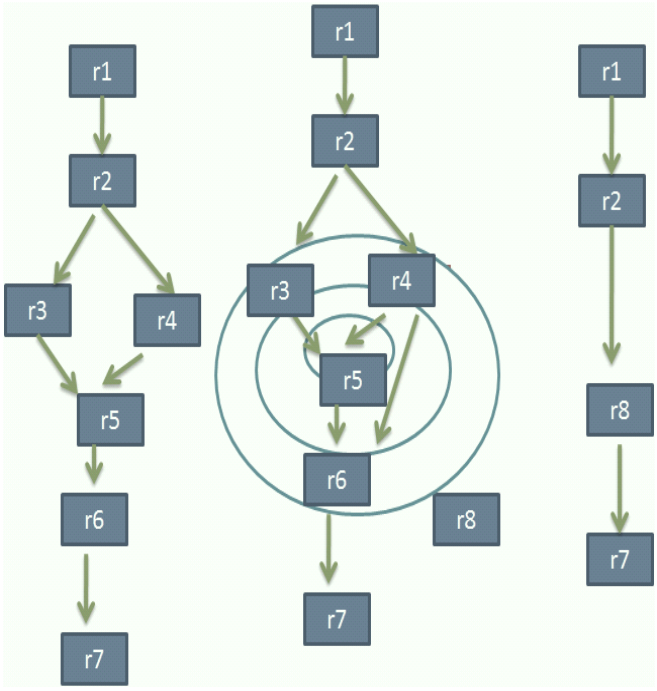

Fig. 27. Memory area implemented with resources Reducible flow mechanism and operational optimal condition

VI. Conclusion

Recently ASIP in our embedded system provides the benefits of flexibility and achieving excellent performances with low power consumption and ASIP also improves the functionality and design complexity with retargateable compiler technology. In real time embedded system designer implements the processor and memory architecture according to our application specific operational probability. ASIP system used the target machine can deploy more sophisticated instructions, which can have the capability to perform specific operations much efficiently for low power embedded system. Compilers and profiling mechanisms are also analyzed for ASIP and implements memory area reduction technique which improve the application execution performance. An effective cycle time, delay and scheduling prediction mechanism is used for memory implementation. An Efficient latency prediction technique is designed for operational serialization with the help of profiler and application specific computational complexity analyzed according to profiling execution delay time which is used in various high performance embedded devices.

References

[1] P. R. Panda, Nikhil D. Dutt "Data Memory Organization and Optimization in Application Specific Systems," In *Proceedings of the IEEE design & tests of Computers,(2001).*pp.56-58.

[2] M. K. Jain, M. Balakrishnan and A. Kumar "Integrated on-chip storage evaluation in ASIP synthesis," In *Proceedings of the 18th International conference on VLSI design (2005).*pp. 274-279. DOI**:** http://dx.doi.org/10.1109/ICVD.2005.112

[3] P. Meloni, S.Pomata, G. Tuveri, S. Secchi. L. Raffo, M. Lindwer. "Enabling fast ASIP design space exploration: An FPGA based runtime reconfigurable prototype," Hindawi Publication Cooperation, J. VLSI design (2012).

[4] Z. Prikryl, J.Kroustek, T. Hruska, D. Kolar. "Fast just in time translated simulator for ASIP," IEEE 14TH International symposium on Design and

diagnostics of electronic circuits and system (DDECS), 2011, pp.279-282.

[5] P. Meloni., S. Pomata, L. Raffo, M. Lindwer "Combining on-hardware prototyping and high level simulation for DSE of multi-ASIP system," IEEE Embedded Computer Systems (SAMOS), 2012, pp.310- 317.

[6] L. T. Clark, E. J. Hoffman, J. Miller, M. Biyani, Y. Liao, S. Strazdus, M. Morrow, K.E. Velarde and M. A. Yarch. "An Embedded 32-b Microprocessors core for low-power and high performance applications," IEEE J. of Solid-State Circuits 36(11), 2001, pp.1599-1608.

[7] E. Diken, R. Jordans, H. Corporaal "Build master: efficient ASIP architecture exploration through compilation and simulation result caching," IEEE 17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, 2014, pp. 83-88.

[8] E. Diken, R.Jordans. R. Corvino, H. Corporaal, F.A. Chies. 2014. "Construction and exploration of VLIW ASIPs with Heterogeneous vector-width," J. of Microprocessor and Microsystem. 2014. Vol.18, no. 8,pp.947-959.

[9] M. F. Jacome, G. D. Veeciana "Lower bound on latency of VLIW ASIP data paths," *International conference on computer aided design.IEEE*.1999, pp. 261-269.

[10] K. Karuri, R. Leupers, "Fine grained application source code profiling for ASIP design," *In the proceeding of 42nd design automation conferences.2005*.pp.329-334.

[11] X. Li, W. Zhou, D. Liu. "Application source code profiling for ASIP memory subsystem design," Procedia engineering, 2012, vol. 29, pp..3160-3164.

[12] A. Hoffmann. T. Kogel, A. Nohl. S. O. Brarun,O. Wahlen, A. Weiferink, and H. Meryr "A Novel Methodology for the Design of Application Specific Instruction-Set Processor Using a Machine Description Language". IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems (TCAD*)*, 20, 11 2001. pp.1338-1354.

[13] Z. Prikryl "Fast simulation of pipeline in ASIP simulator," IEEE International 14th workshop on microprocessor test and verification, 2014.pp.10-15.

[14] H. M. Hassan**,** K. Mohammed and A.F. Shalash. "Implementation of a reconfigurable ASIP for high throughput low power DFT/DCT/FIR engine," EURASIP J. on Embedded Systems, 2012.

[15] R. Leupers. "LANCE: A C Compiler Platform for Embedded Processors. Embedded System/Embedded Intelligence," Feb 2001.

[16] P.D. O. Castro, C. Akel, E. Petit, M. Popov, W. Jalby "CERE: LLVM based Codelet Extarctor and REplayer for Piecewise Benchmarking and Optimization," J. ACM Transactions on Architecture and Code Optimization (TACO), 2015. DOI**:** http://dx.doi.org/10.1145/2724717

[17] A. Mathur, M. Fujita, E. Clarke and P. Urard "Functional equivalence verification tools in High level synthesis flows," IEEE design and test of computer,vol. 26,no. 4 2009,pp. 88-95.

[18] P. J. Pingree, L. J. Scharenbroich, T. A. Werne and C. Hartzell "Implementing legacy-C algorithm in FPGA co-processors for performance accelerated smart payloads," *In proceeding of Aerospace conference*, 2008, pp.1-8.

[19] A. Putnam, S. Eggers, D. Bennett, E. Dellinger, J. Mason, H. Styles, P. Sundararajan and R.Witting "Performance and power of cache-based reconfigurable computing," *In Proceeding of ISCA' 09,* 2009. pp.395-405.

[20] J. Zhang, Z. Zhang., S. Zhou, M. Tan, X. Liu, X. Cheng, and J. Con.. Bit-level optimization for high level synthesis and FPGA-based acceleration. *In proceeding of FPGA'10,*pp. 59-68.