Research Paper Engineering

Development of Tool for Data De-Serialization of Memory Dumps (Octeon & TI DSP Multicore Environment)

***Sumukha Prasad U** Digital Communication Engineering, RV College of Engineering, Bengaluru, India. *Corresponding Author

Giri Prasad Nokia Networks, R&D Technology Centre, Bengaluru, India.

RK Manjunath Digital Communication Engineering, RV College of Engineering, Bengaluru, India.

ABSTRACT

In the Digital Era, the Wireless Mobile Networks are more predominant which are wide spread in all the applications and the robust designs are being prepared for it, the end-user has noticed and there is a need for faster, accurate and much more capable versions of Base Station Controllers to uphold the highly dense cellular traffic. The best methodologies are being usage of high speed multi-core processors and using multi-threading concepts. Although the chip accuracy and speed have boosted to a greater extent, multi-core processing remains as the favorite concept for all the electronic manufacturers including the mobile handset makers. Over the years, even though adding of multiple cores to the DSPs, full benefit & efficiency of the technology is not exploited. This paper deals with the cause for the hindrance that is causing the multi-core processors from giving its fullest thrust in its functionality. The main reason is due to unavailability of its register contents during the crash of a processor. The BSC2000 uses multicore Octeon & Texas Instruments DSP in its ETP interface cards; henceforth this paper provides a methodology to retrieve the register contents from processor's crash binary contents. A standalone tool is developed to provide the contents in the readable format and displayed according to the processor contents.

KEYWORDS Octeon Processor, Tomahawk TI DSP, Base Station Subsystem 2000, Multi Core Processors, Transcoder Rate Adapter Unit, Transcoder Sub-Multiplexer Unit, DSP Manager, Small Form Factor Plug-in Module.

I. INTRODUCTION

The integral part of Telecommunication subsystem, Base Station Subsystem (BSS) is an important module under digital cellular system which contains Base Station Transceiver System (BTS) and Base Station Controller (BSC) to provide radio coverage and handles control functions in the designated geographical area. The BTS includes RF Transceiver and other Base Band Hardware sub-systems required to establish communication with the Mobile equipment. The BTS provides and RF coverage over a specific area known as Cells, BSC would control many such cell sites by intelligent control functions via BTS that are attached to it. These BTS's are interconnected with BSC through an Abis-Interface and the same BSC is extended with the connection to MGW/MSC via A-Interface. Packet Abis and A-interface over IP features require an Exchange Terminal for Packet transport (ETP) plug-in unit (PIU) to BSC and TCSM2000. There are two hardware variants of ETP PIU depending on its positioning in the BSS. The ETP PIU for Packet Abis is positioned on the Abis edge of the BSS and ETP-A PIU is placed on the A-interface edge. On the Abis-edge, depending on the connectivity with BCF, ETP PIU can be further categorized into; ETPT (TDM connectivity with BCF) and ETPE (Ethernet connectivity with BCF). On the A-interface-edge, depending on the placement of the Transcoder, ETP PIU can be categorized into the following variants; ETPC in Combi TCSM (Transcoder in BSS), ETPC in Standalone TCSM (Transcoder in BSS) and ETPA (Transcoder in Media Gateway).

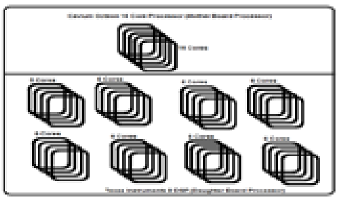

Cavium Octeon 16 Core Processor (Mother Board Processor)
16 Cores
6 Cores
Texas Instruments 8 DSP (Daughter Board Processors)

Figure 1: Blackbox Core Representation.

BSC has many such hardware ETP cards which is considered as Blackbox in the further references, which has a mother board consisting of a single Octeon Processor (16 Cores) and the daughter board has 8 TI DSPs (each having 6 Cores) (Refer Fig 1).

II. MOTIVATION

There is no existing tool for converting the memory dumps of DSP into readable format, the requirement for Nokia is to develop the standalone tool, which should be flexible (making changes should be easy), generic (easily understandable for other development engineers) and robust. The developed tool can be enhanced to support memory dumps from Octeon Processor also.

III. THE APPROACH

A. Blackbox Plug in Unit:

Blackbox plug-in unit in DX200 platform provides BSC2000 network element with Abis-over-IP-over-Ethernet interface (type E) and Abis-over-IP-over-TDM interface (type T) towards IP BTS's. The PIU is responsible for IP processing, TRAU frame handling and TDM conversion for interfacing with the BSC2000 switching network (GSW). Blackbox is able to convert up to 8000 CS channels or up to 128Mbit/s PS traffic from Ater/TRAU to packet Abis packets and vice versa. IP over Ethernet: Blackbox provides two 1Gbit/s ethernet ports towards BTS's, for connection to active and redundant site switch. Both optical and copper interfaces are featured by hot pluggable SFP modules which are not part of blackbox PIU's structure. IP over TDM: Blackbox provides two Hotlink interfaces that are directly linked to an Blackbox2 (half) of the BSC2000 in order to establish the connection with BTS's. 1+1 card redundancy protection scheme is adopted for the blackbox.

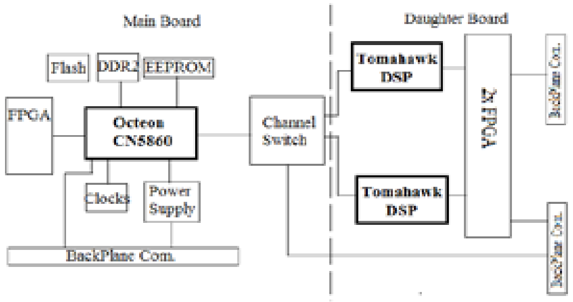


Figure 2: Blackbox Hardware Layout.

The Cavium Octeon multi-core processor has sixteen cnMIPS cores on a single chip (Refer Fig 1 & Fig 2). These cores work with a 600 MHz operating clock. The Octeon packet processor acts as IP message forwarder and router. Packet Processor performs parameter and format translation and routes the packets between the network interfaces.

The DSP block consists of eight (8) Texas Instruments TMS320TCI6486 Tomahawk multicore DSP DSPs. There are 6 cores@ 625 MHz per DSP. DSP blocks on the blackbox PIU are used for converting IP/RTP packet data to legacy CS TRAU frames and to support legacy PS connectivity over HDLC with the PCU cards.

The number of DSPs populated on the PIU depends on the blackbox variant. When any of the Blackbox's Octeon core group (Uplink core group / Downlink core group / Management core) is highly overloaded (i.e. High Overload Limit exceeded), Blackbox will raise 'blackbox_overload_condition_in_octeon' alarm. Blackbox will cancel the alarm when the high overload condition has gone away from all the Octeon core groups. Blackbox will raise 'blackbox_overload_condition_in_dsp' alarm when any of the DSP core is highly overloaded. Blackbox will raise this alarm only once and cancel the alarm when the CPU load of all DSP cores goes below high overload limit. The PCU-IDs (consisting of BCSU logical address and PCU index) that are affected due to high overload along with the count of blocked HDLC links in that DSP core will be set in 'blackbox_overload_condition_in_dsp" alarm. While cancelling the same alarm, number of HDLC links that were blocked during the overall overload situation will be updated.

B. Implementation Description:

The default action of certain signals is to cause a process to terminate and produce a core dump file and a disk file containing an image of the process's memory at the time of termination. This image can be used in a debugger to inspect the state of the program at the time that it terminated. Blackbox memory logs are also essential for debugging the memory allocations of each memory cards. These DSP`s when crashed, they leave behind a simple representation of crash information, which is available in the blackbox for debugging. The black-box utility provides information about the last known snapshot of process states, whenever any Octeon or DSP process suffers a failure. This information is used to diagnose the cause of failure. The black-box data for Octeon is saved in a pre-designated area in the flash mounted on Linux file system (henceforth referred to as Linux flash file system (Linux FFS)). It is expected that the fault data provide pointers to the cause of the component failure. Failure causes can be bus errors, divide-by-zero, invalid memory access etc.

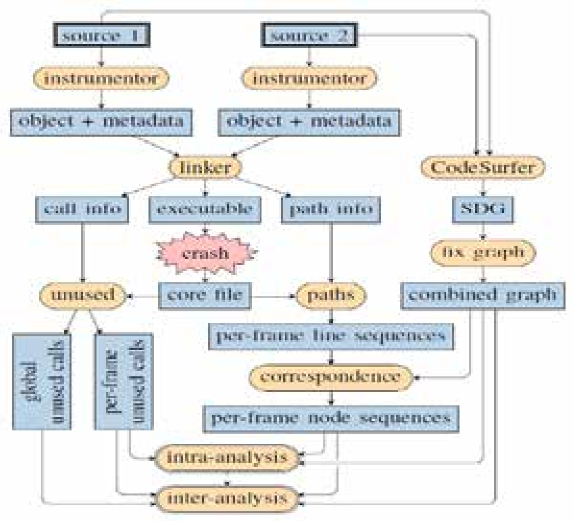


Figure 3: Memory Dump Analysis.

The black box for Octeon cores is built from the error handler which captures all the relevant information and stores it in Octeon's shared memory. When Simple Executive goes down, the Blackbox module takes the dump from shared memory and stores it in the Linux FFS. The black box for management core uses core dump utility provided by the Linux kernel. Whenever any thread or process crashes, the core dump is created by the kernel. Along with core dump, blackbox for Linux core is also created which contains stack dump, register set etc. To enable this feature, changes have been made in the kernel source files. Whenever a process on DSP core crashes, the dump of its stack, registers etc are stored in the form of black box. DSP core sends the black box data to Octeon and then Octeon resets the DSP (all Cores). If this core crashes again, then black box data is not overwritten and is placed in a different location thus multiple black box of a single core of a particular DSP is stored.

A tool is required to debug this information and de-serialization from memory dumps of Octeon and Tomahawk DSP (TMS320CI6486) multicore environment for the analysis of DSP crashes. The memory dump collected from the blackbox; the tool scrutinizes the dumps and converts it into corresponding data structures (Refer Fig 3). This helps to understand the scenario for the crash of the processor and the cause of its happening. Crash dump analysis is the ability to record the state of the system when a crash occurs and then analyze that state at a later time to determine the cause of the failure. For instance, the state of the stack may be collected in order to generate a call stack showing the calls leading up to the failure. This may be necessary in a production environment where a JTAG connection cannot be made to the live system to debug it, but where problems may still occur and must be analyzed and fixed.

C. Practical Implementation:

Step1: Data Acquisition

Collecting the Binary DSP Dump (>500MB), DSP MAP File, DSP Image File are the primary goals for development of the tool and the memory dump reconstruction.

Step 2: Conversion to Hexadecimal Format

The Binary DSP dump file which is in non-readable format is converted into a Hexadecimal format for good understanding of the indices of variables and structures.

Step 3: Feature Extraction of MAP File & Mapping of Binary

The MAP file has the details of variable mappings (actual indices), which are to be extracted into tool with absolute indices. Based on the absolute addressing the range of binary data is mapped to the corresponding variables.

Step 4: Classification of Variables into 9 Sub-Modules
All the absolute indexed variables are segregated into 9 Sub-Modules (9 Processor's Memory Address Range) based on the starting address and are displayed as the expected outcome of the tool.

Step 5: Structure Extraction based on C Parser
Since Processor coding is done using C programming, the C source file parser which is named as Convert::Binary::C module is utilized from the Perl library for Byte wise extraction of contents for the C typedef Structures.

Step 6: Integrated Standalone Tool Development
An Integrated Standalone tool is developed for displaying the extracted Variable and Structure features with the Choice of DSP or Octeon Processors. The tool can get the data crunching done in a matter of seconds.

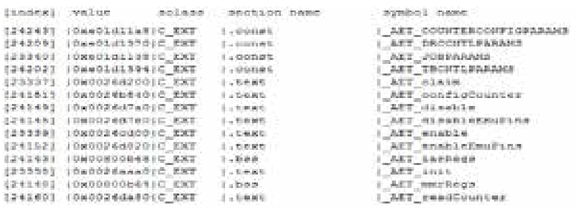
D. Binary File:

Binary file is not a text file; it can contain any type of data which is encoded in binary form for storage & processing purposes in computers (Refer Fig 4). Binary files are usually thought of as being a sequence of bytes, which means the binary digits (bits) are grouped in eights. Binary files typically contain bytes that are intended to be interpreted as something other than text characters. Compiled computer programs are typical examples; indeed, compiled applications are sometimes referred to, particularly by programmers, as binaries. But binary files can also mean that they contain images, sounds, compressed versions of other files, etc., in short, any type of file content whatsoever. Some binary files contain headers, blocks of metadata used by a computer program to interpret the data in the file. The header often contains a signature or magic number which can identify the format.

Figure 4: Blackbox-DSP Image file in '.out' format (Binary).

E. Image File:

Image files extraction from the black box, which contains the global variable declarations along with its values being indexed in the non readable (binary) format as shown in the Fig 4. This non-readable (binary) format is converted into readable shadow file for the analysis as shown in the Fig 5. This image file is primary requirement for analyzing the DSP dump.



```
[index]  value        sclass   section name        symbol name

[24243]  |0xe01d11a8|C_EXT    |.const              |_AET_COUNTERCONFIGPARAMS
[24209]  |0xe01d1370|C_EXT    |.const              |_AET_DRCCNTLPARAMS
[23340]  |0xe01d1138|C_EXT    |.const              |_AET_JOBPARAMS
[24202]  |0xe01d1394|C_EXT    |.const              |_AET_TBCNTLPARAMS
[23337]  |0x00268200|C_EXT    |.text               |_AET_claim
[24181]  |0x0024b640|C_EXT    |.text               |_AET_configCounter
[24149]  |0x002647a0|C_EXT    |.text               |_AET_disable
[24145]  |0x002687e0|C_EXT    |.text               |_AET_disableEmuPins
[23338]  |0x0024cd00|C_EXT    |.text               |_AET_enable
[24152]  |0x0026d020|C_EXT    |.text               |_AET_enableEmuPins
[24143]  |0x00800b48|C_EXT    |.bss                |_AET_isrRegs
[23358]  |0x0024aaa0|C_EXT    |.text               |_AET_init
[24140]  |0x00800b48|C_EXT    |.bss                |_AET_mmrRegs
[24180]  |0x0026da80|C_EXT    |.text               |_AET_readCounter
```

Figure 5: Blackbox-DSP Image file into converted '.txt' format.

F. Binary DSP Dump:

Binary DSP Dump contains the entire contents of the physical memory at the time of the crash. This type of dump will require that there is a page file at least the size of physical memory plus 1MB (for the header). Because of the page file requirement, this is an uncommon setting especially for systems with large amounts of RAM. In computing, a core dump, memory dump, or system dump consists of the recorded state of the working memory of a computer program at a specific time, generally when the program has terminated abnormally (crashed). In practice, other key pieces of program state are usually dumped at the same time, including the processor registers, which may include the program counter and stack pointer, memory management information, and other processor and operating system flags and information. Core dumps are often used to assist in diagnosing and debugging errors in the programs. Binary DSP Dump is extracted from the Black box which is in binary format as shown in Fig 6.

Figure 6: Blackbox-DSP Dump File (Binary format).

G. Hex Dump File:

In computing, a hex dump is a hexadecimal view of computer data, from a storage device. Looking at a hex dump of data is commonly done as a part of debugging, or of reverse engineering. In a hex dump, each byte (8-bits) is represented as a two-digit hexadecimal number. Hex dumps are commonly organized into rows of 8 or 16 bytes, sometimes separated by whitespaces. Some hex dumps have the hexadecimal memory address at the beginning and/or a checksum byte at the end of each line. The Binary DSP Dump which is converted to HEX Dump, clearly memory address can be observed on the left side and the checksum byte on the extreme right in the Fig 7.

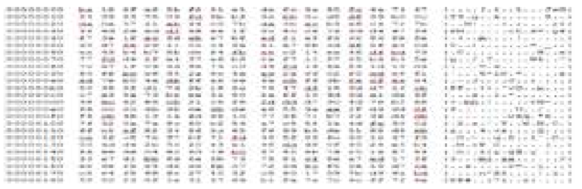
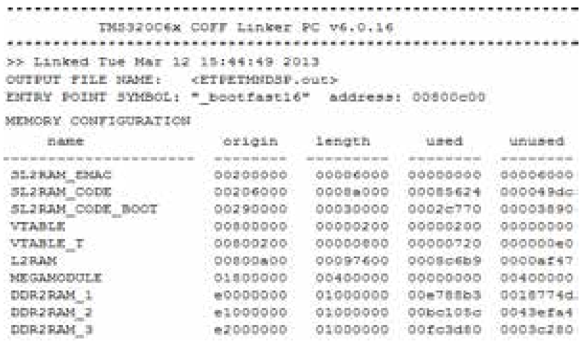


Figure 7: Blackbox-HEX Dump File.

H. MAP File:

Map file is also known as a linker file which has the details of global variables with its memory content length along with its address as shown in Fig 8. These are typically plain text files that indicate the relative offsets of functions for a given version of a compiled binary. It defines the relationships between objects, points map server to where data are located and defines how things are to be considered for decoding.



```
******************************************************************
           TMS320C6x COFF Linker PC v6.0.16
******************************************************************
>> Linked Tue Mar 12 15:44:49 2013
OUTPUT FILE NAME:   <ETPETMNDSP.out>
ENTRY POINT SYMBOL: "_bootfast16"  address: 00800c00
MEMORY CONFIGURATION
         name                origin    length      used     unused
--------------------------  --------  ---------  --------  --------
  SL2RAM_EMAC               00200000   00006000  00000000  00006000
  SL2RAM_CODE               00206000   0008a000  00085624  000049dc
  SL2RAM_CODE_BOOT          00290000   00030000  0002c770  00003890
  VTABLE                    00800000   00000200  00000200  00000000
  VTABLE_T                  00800200   00000800  00000720  000000e0
  L2RAM                     00800a00   00097600  0008c6b9  0000af47
  MEGAMODULE                01800000   00400000  00000000  00400000
  DDR2RAM_1                 e0000000   01000000  00e788b3  0018774d
  DDR2RAM_2                 e1000000   01000000  00bc105c  0043efa4
  DDR2RAM_3                 e2000000   01000000  00fc3d80  0003c280
```

Figure 8: Blackbox-DSP MAP File.

I. Memory Blocks Allocation:

Memory Blocks of a processor are referred as fixed-size block allocations, since they undergo fragmentation because of variable block sizes and it is not advisable to use them in real time system due to performance. The best way is to predefine the number of memory blocks for the variables. The Processor can allocate access and release the free blocks at the runtime. DSP Dumps are sent from DSPM to the Octeon in an 8 array chunks as shown in Table 1 and the 9th block is the memory chunk of the Octeon itself.

Table 1: Memory Mapping Table.

Actual Address Range	Absolute Address Range
0x00200000 --> 0x002C0000 0x02C80000 --> 0x02C84000 0x10800000 --> 0x10898000 0x11800000 --> 0x11898000 0x12800000 --> 0x12898000 0x13800000 --> 0x13898000 0x14800000 --> 0x14898000 0x15800000 --> 0x15898000 0xE0000000 --> 0xE6000000	0x00000000 --> 0x000C0000 0x000C0000 --> 0x000C4000 0x000C4000 --> 0x0015C000 0x0015C000 --> 0x001F4000 0x001F4000 --> 0x0028C000 0x0028C000 --> 0x00324000 0x00324000 --> 0x003BC000 0x003BC000 --> 0x00454000 0x00454000 --> 0x06454000

J. Mapping of Variables:

Actual Address of the variable: AV

Actual Starting Address: AS	Absolute Starting Address: ABS
Actual Ending Address: AE	Absolute Ending Address: ABE

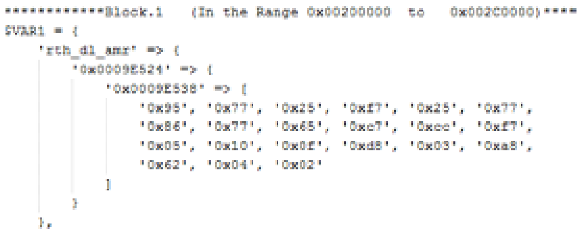
OFFSET = (AV – AS) + ABS

'rth_dl_amr': (In the Range 0x00200000 to 0x002C0000)	OFFSET = (0x0029E524 - 0x00200000) + 0x00000000 which is: 0x0009E524. The location IN THE FILE where the variable starts is: 0x0009E524.

AV = 0x0029E524.
AS = 0x00200000.
	AE = 0x002C0000.
	ABS = 0x00000000.

Table 2: Variable Memory Mapping with Example.

```
Var1: 'rth_dl_amr' => {

'0x0009E524' => {

'0x0009E538' => [

'0x95', '0x77', '0x25', '0xf7', '0x25', '0x77',

'0x86', '0x77', '0x65', '0xc7', '0xcc', '0xf7',

            '0x05', '0x10', '0x0f', '0xd8', '0x03', '0xa8',

            '0x62', '0x04', '0x02'

          ]

        }

    },
```

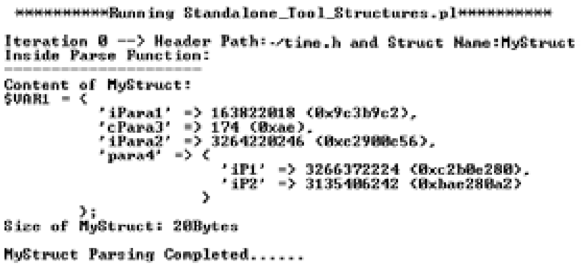


```
*************Block.1   (In the Range 0x00200000  to   0x002C0000)****
$VAR1 = {
    'rth_dl_amr' => {
        '0x0009E524' => {
            '0x0009E538' => [
                '0x95', '0x77', '0x25', '0xf7', '0x25', '0x77',
                '0x86', '0x77', '0x65', '0xc7', '0xcc', '0xf7',
                '0x05', '0x10', '0x0f', '0xd8', '0x03', '0xa8',
                '0x62', '0x04', '0x02'
            ]
        }
    },
```

Figure 9: Variables with its Contents extracted from DSP Dumps.

From Fig 9 & Fig 10, it is evident that the variables & structures are mapped onto its content from the DSP dump; this is an example for exact reading of memory address with specific indexing address of variables from the map file along with its structures defined.

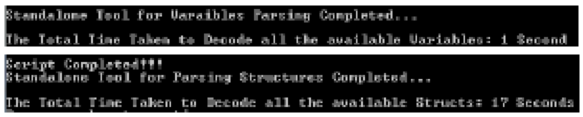
There is a huge amount of data packet transactions happening in the real communication interfaces, hence the offline data analysis plays a major role in understanding the causalities or the error happened during the live channel traffic on the BSC interfaces. By rectifying the error happened at the incidental moment, by retrieving the lost data content at that moment recorded in the crash logs of the processors, will resolve the problem and also prevents the future happenings of the same type of errors. The manual effort (normally requires more effort and man hours) required to analyze the dump analysis is taken care off within 17 seconds of computational accuracy (Fig 11) and recovery of the abstracted encapsulated data from the crash dumps.



```
 ***********Running Standalone_Tool_Structures.pl***********

Iteration 0 --> Header Path:-/time.h and Struct Name:MyStruct
Inside Parse Function:
============================
Content of MyStruct:
$VAR1 = {
          'iPara1' => 163822018 (0x9c3b9c2),
          'cPara3' => 174 (0xae),
          'iPara2' => 3264220246 (0xc2900e56),
          'para4' => {
                       'iP1' => 3266372224 (0xc2b0e280),
                       'iP2' => 3135406242 (0xbae280a2)
                     }
        };
Size of MyStruct: 20Bytes

MyStruct Parsing Completed......
```

Figure 10: Structures with its Contents extracted from DSP Dumps.

This tool developed to extract the variables and structures with its contents followed an algorithm, to fetch the required data from various types of dumps and binary files which are available from the black box. At the time of processor crash the final glimpse of the data is captured and the data report is being sent to the black box by the DSP manager which is coordinated by the Octeon. By using this tool, on successful reconstruction of data, the memory content at the time of processor crash can be traced out and analyzed for its functionalities. The standalone tool is flexible and robust developed using C & Perl Scripting; the methodology used to decode or de-serialize of the variables and structures of the Octeon & DSP Crash dumps are reconstructed successfully.



```
Standalone Tool for Varaibles Parsing Completed...

The Total Time Taken to Decode all the available Variables: 1 Second
```

```
Script Completed!!!
Standalone Tool for Parsing Structures Completed...

The Total Time Taken to Decode all the available Structs: 17 Seconds
```

Figure 11: Quantitative Time Measurement of the Tool.

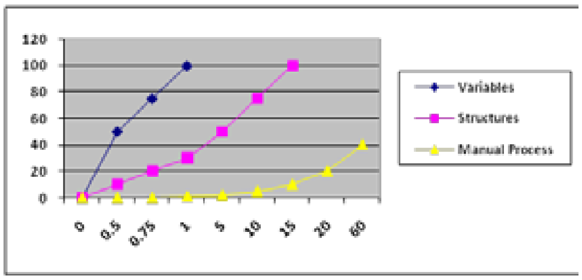


Figure 12: Time v/s Completion rate for Tool (Variables & Structures) and Manual Process.

IV. OUTCOME & CONCLUSION

The effective manner of implementation is being estimated in terms of efforts for the computation process, the graphs (Fig 12, 13 & 14) clearly indicate the reduction in the manual effort greater than 50% in each level of procedures. In Fig 12 & Fig 13, there is an unimaginable difference in the manual & tool efforts which accounts for robust computation accuracy which is possible through the standalone tool. The amount of time is saved and therefore the large data traffic handled by the processors is being analyzed at faster rate. This offline analysis gives an exact scenario of register contents extracted from the binary DSP crash dump. Hence it would become easy for the BSC to take care of register and its contents in the future handling. The Standalone tool is successfully implemented and tested on the Linux environment for its accuracy and robustness. The future scope of implementation for this concept would be to enhance the methodology for online data analysis and the analysis report is being provided instantly for the processor to take into fractional accuracy in the data crunching, which shall avoid the repetition in processor's failures.

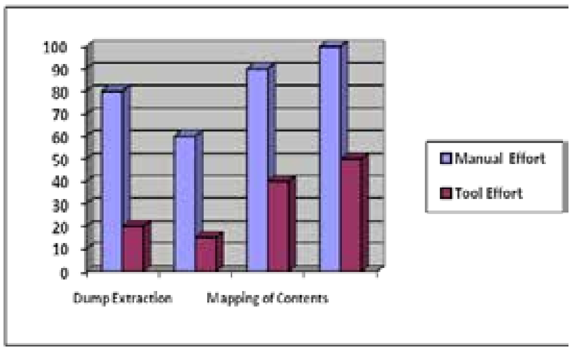


Figure 13: Bar Chart indicating Process for Tool and Manual Effort w.r.t its Completion Slab.

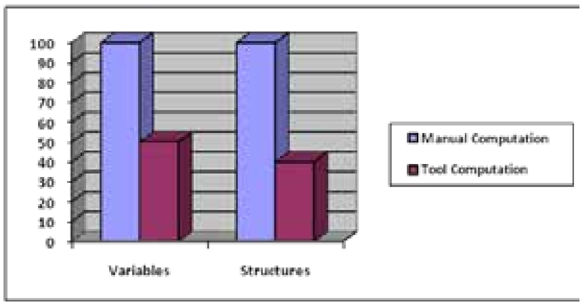


Figure 14: Bar Graph depicting the clear comparisons between Variable & Structure extraction by Tool & Manual Computation.

V. ACKNOWLEDGEMENTS
Sincere regards to Mr.Shrinidhi GV, Mr.Mohan Kumar C and Mr.Lijin Shyam K of Nokia Networks & RV College of Engineering for providing all the necessary guidance, support and facilities for carrying out this project successfully.

REFERENCES

[1] Liu Guangqi, Wang Lianhai, Zhang Shuhui, Xu Shujiang and Zhang Lei, 'Memory Dump and Forensic Analysis Based on Virtual Machine', Proceedings of 2014 IEEE International Conference on Mechatronics and Automation, pp. 1773-1777 August 2014. | [2] Abdullah Kayi, Olivier Serres and Tarek El-Ghazawi, 'Adaptive Cache Coherence Mechanisms with Producer–Consumer Sharing Optimization for Chip Multiprocessors', IEEE Transactions on Computers, Vol.64, No.2, pp.316-328, February 2015. | [3] Hao yu-jie, Liu tao, 'Building a robust packet control unit with network processors', IEEE 2015. | [4] Haiying Shen, Ze Li and Kang Chen, 'A Scalable and Mobility-Resilient Data Search System for Large-Scale Mobile Wireless Networks', IEEE Transactions on Parallel & Distributed Systems,Vol.25,No.5, pp.1124-1134, May 2014. | [5] Gita Shah, Annappa and K. C. Shet, 'Efficient Way of Searching Data in MapReduce Paradigm', International Conference on Computing for Sustainable Global Development, pp.305-310, IEEE 2014. | [6] Amrit Pal and Sanjay Agrawal, 'An Experimental Approach Towards Big Data for Analyzing Memory Utilization on a Hadoop cluster using HDFS and MapReduce', First International Conference on Networks & Soft Computing, pp.442-447, IEEE 2014. | [7] Paolo Burgio, Giuseppe Tagliavini, Francesco Conti, Andrea Marongiu and Luca Benini, 'Tightly-Coupled Hardware Support to Dynamic Parallelism Acceleration in Embedded Shared Memory Clusters', EDAA 2014. | [8] Jingjing Wang, Deepak Jagtap, Nael Abu-Ghazaleh and Dmitry Ponomarev, 'Parallel Discrete Event Simulation for Multi-Core Systems: Analysis and Optimization', IEEE Transactions on Parallel & Distributed Systems, Vol.25, No.6, pp.1574-1584, June 2014. | [9] Baohua Yang, Jeffrey Fong, Weirong Jiang, Yibo Xue and Jun Li, 'Practical Multituple Packet Classification Using Dynamic Discrete Bit Selection', IEEE Transactions on Computers, Vol. 63, No. 2, pp.424-434, February 2014. | [10] Shih-Kun Huang, Min-Hsiang Huang, Po-Yen Huang, Han-Lin Lu and Chung-Wei Lai, 'Software Crash Analysis for Automatic Exploit Generation on Binary Programs', IEEE Transactions on Reliability, Vol.63, No.1, pp.270-289, 2014. | [11] Jeroen van den Bos, 'Lightweight Runtime Reverse Engineering of Binary File Format Variants', CSMR-WCRE ('14), pp.367-370, IEEE 2014. | [12] Zhou Lei, ZhaoXin Li, Yu Lei, YanLing Bi, Luokai Hu and Wenfeng Shen, 'An Improved Image File Storage Method Using Data Reduplication', IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, pp.638-643, 2014. | [13] Jeremias R¨oßler, Andreas Zeller, Gordon Fraser, Cristian Zamfir, George Candea, 'Reconstructing Core Dumps', IEEE Sixth International Conference on Software Testing, Verification and Validation, pp.114-123, 2013. | [14] Chih-Hung Chang, Chih-Wei Lu, William C. Chu, Pao-Ann Hsiung, Nien-Lin Hsueh, Chorng-Shiuh Koong, Chao-Tung Yang, 'An Integrated Development Environment to Support the Multi-core Embedded Systems Development',12th International Conference on Quality Software, pp.258-264, IEEE 2012. |