International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:268
Vol.2, No.11, November 2013 DOI:10.15693/ijaist/2013.v2i11.9-13

Security Application using multiprocessors firmware architecture

A.L.SRIRAM, SUBRAMANIAN, SWARNA SWEETY

A.P. III, SASTRA University,Tanjavur.
Project Manager, Mistral solutions.Banglore.

Abstract

Modern day defence electronic systems running complex software applications require very huge processing power. These algorithms consume large quantity of data acquired from multiple channels using high-speed ADCs. Many of these systems require data from multiple channels to be captured and processed concurrently for churning out useful information. These systems require enormous memory and high-speed data bus for transfer of data in real-time. These are complemented with processors running at high clock speeds, with specialized signal/vector processing engines. This article will outline a basic approach for designing real-time applications using these multi processor boards.

Introduction

In this golden era of shrinking integrated electronic circuits, the scope of building powerful systems, in small form factors is becoming feasible.[1] Concern regarding the weight and size of defence electronics systems has driven the designers to look for processing boards, of smaller form factors with very high processing power. This has led to evolution of board design, especially boards with multiple processors to meet the demand of these high computation intensive applications.

There are many multiprocessor COTS boards available in the market that specifically target defence applications. Few of these boards provide a platform, with symmetric architecture, and inherent support for high-speed data acquisition PMC/XMC/FMC daughter cards. These boards are available majorly on VME or VPX backplane. Depending on the complexity of the system, multiple boards can be used, with seamless data communication fabric between the multiple processors and across multiple boards over the backplane.

Operating system

Complex defense electronics systems require real time operating systems for multitasking.[2] Since most of these systems are bound to be onboard military aircraft, or ships, they have to undergo recommended certification before deployment. This warrants the use of well proven, certified or certifiable operating systems. Few important things to be considered are the OS tick, foot print, latencies, context switch time, flexible scheduling and other aspects such as availability of development tools, required software stacks and the BSP.

The board vendors often provide sophisticated software libraries for vector signal processing, inter node communications etc.[3] The functions required though for the system has to be thoroughly profiled, while arriving at the system time budget.

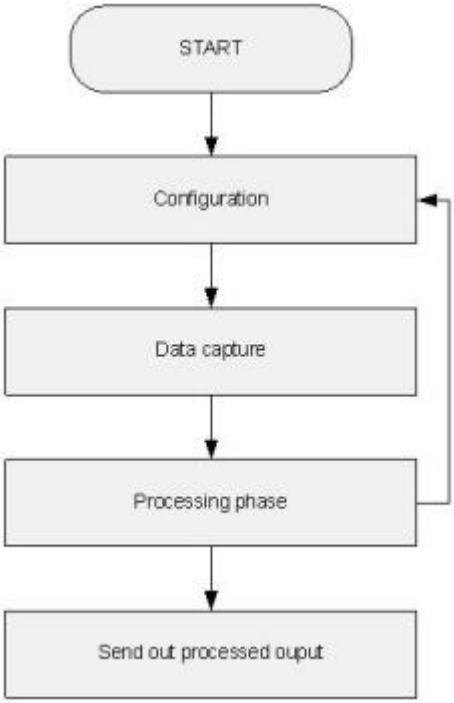
Design philosophy

We will now discuss a structured design methodology for multiprocessor application design. [4]

Architecture

The architect needs to perform top level mental modelling of the system functionality before translating it into feasible architecture. Various parameters of the processing boards have to be thoroughly examined and tradeoffs have to be weighed before arriving at the best suitable architecture. Various possible architectures have to be studied before zeroing in on the most efficient one for the application. [5]

The design of a multi processor application follows the usual software design process. The architecture outlines the software modules and the hierarchy of modules. Detailed design follows the architecture, addressing much finer aspects of each module, to the level of each data element.



Design

Start off with a context diagram, in which the entire system with all the external interfaces is visualized. Then the context diagram is broken down to multiple levels to include finer details of the design. This also involves the modular decomposition. The entire functionality of the system is broken down into modules, each addressing a specific functionality. This brings a structure to the application. A good layered approach for the modular decomposition will guarantee scalability and configurability.

The control and data coupling between the modules has to be defined properly. The Control flow Diagrams (CFDs) and Data Flow Diagrams (DFDs) would help to get much clarity on the design.[6]

Create a Main module, which will co-ordinate and control all the other identified modules, to provide the required system functionality. The intended operation of the system has to be broken down into phases, such as configuration phase, data acquisition phase, processing phase, presentation phase etc. The performance requirement to be met by the identified modules mapping to the phases of operation shall be validated.

Budget the time required for all the phases to achieve the required system performance. This step will help identify the critical modules, in terms of performance and memory usage. All the software libraries including the signal processing library, and the inter node communication libraries have to be very well profiled, in a real-like scenario. This helps eliminate unpleasant surprises at a later stage of development.

Now the job of spreading the application on multiple processing nodes starts. The system architecture should take care of the load balancing between the various processing nodes. The application architecture can group the nodes into the controlling nodes and the processing nodes. The controlling nodes shall utilize the full power available with the processing nodes, by parallel loading.

The software running on multiple processors are doing actual parallel processing, rather than multi threaded applications running in a single core.

Analyze the resource requirements of each module for the optimized performance, in terms of processor time, quantity of input and output data, memory and identify the critical modules. It is the critical modules that have to be designed to run parallel on multiple nodes. These critical modules are usually the modules associated with the processing of the captured data.

Complex algorithms require large amount of memory, which may be practically impossible to provide in a single processor in a useful manner. Splitting and spreading the algorithm on multiple cores or processors helps to efficiently use the memory available over multiple processors.[7]

The critical processing modules require the input data to be supplied at or faster than the capture rate. Certain systems would have multiple data acquisition cards, controlled by different processors. The data captured by all the data acquisition modules have to be made available for processing. This requires transfer of enormous quantity of data across the processors. This has to be very well

considered while placing the processing modules on multiple nodes.[8]

Apart from the input and output data there are many other messages flowing between all the processing nodes. These messages have to be short and precise. The number of processing nodes, the modules to coexist on a node and the number of tasks on each module shall be determined, using a minimalist approach.

The ease of testing and debugging also has to be kept in mind during the design phase. The access to data at various levels especially during the integration of multiple modules is important for debugging.[9]

Data transfer

The software architecture shall consider the data transfer time between the nodes, and shall reduce it to the maximum extent. An inefficient design involving more data transfers would nullify the advantages of multi processors

The data transfer shall be done in such a way that bus contention between the nodes are eliminated or reduced to the minimum. Though the processing nodes provide multiple DMA channels, they may be using the same physical bus. Initiating multiple channels of DMA for faster data transfer has to be carefully done.

The design should carefully avoid choking of various data transfer channels.

Error handling

The design should consider the state of the system on occurrence of problems at any instant on any remote node. The system should be bound with proper timeout mechanism to bring back the system to normal state, in case of error at any level. The errors occurring at any level should be propagated back, to the top level. The controlling node should have absolute mechanism in place to terminate all the processing nodes and bring them back to normal.

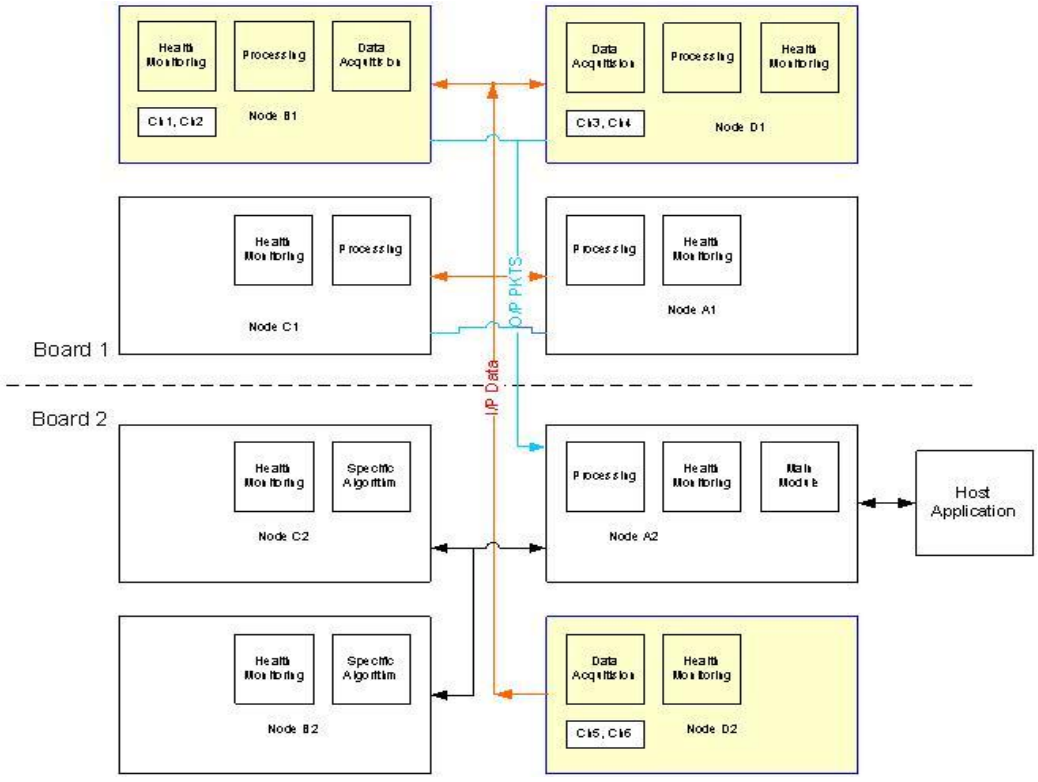
Power dissipation

The heat generated during the peak loading of the processors/FPGAs has to be balanced, to avoid hot spots on the boards. Normally, the chassis housing multiple these boards, have fans (in case of forced air-cooled systems) for the dissipation of heat. The board vendor usually provides software support for determining the temperature at various points on the board.

Health monitoring

The use of multiple processors warrants a very efficient health monitoring, which has to run on all the nodes. The error reporting mechanism has to address the propagation of the error to the top layer on the controlling node.[9]

The various sensors on the board, have to be monitored periodically, and appropriate actions has to be taken on occurrence of any undesired event, for e.g. Temperature rises above the permissible levels, or voltage drop etc.



Case study

This case study showcases the design methodology to build a system, which had 6 channels of data captured at 100MHz. The system had totally 8 processors spread over two boards. There were three data acquisition cards. The system captured and processed about 10MHz on every capture. The performance requirement was to complete the operation on 10MHz data within 1ms.[10]

The major phases of the entire operation were configuration, data acquisition, processing, packetizing and sending over Gigabit Ethernet (GbEth) to other subsystem.

The configuration phase budget was 50 microseconds, and it was not a concern since it will be pipelined, and parallel from the second iteration onwards.

The data capture phase (including the capture and transfer of data to the processing nodes) budget was about 150 microseconds consistently. The data captured at the three capturing nodes (B1, D1 and D2) has to be made available for all the processing nodes. The data transfer was optimized using DMA transfer. The physical bus contention was carefully avoided by initiating non-colliding transfers between the multiple nodes, at any point of time. All the processing nodes were chosen to be on a single board. This was done keeping in mind the data transfer overheads.

The processing phase estimate was around 1.6milliseconds.[12] These estimates were as per the profile figures obtained in a real-like scenario. It was a challenge to spread this critical processing module over multiple nodes. The processing module was spread over four nodes, each processing about 2.5MHz. This reduced the estimate from 1.6milliseconds to approximately 500 microseconds. The number of parallel processing nodes was restricted to four (A1, B1, C1 and D1), since each board had four nodes, and increasing the processing nodes further did not improve the total performance.

The memory requirement of the processing module for processing the entire 10MHz was approximately 23MB. The requirement came to about 6MB when the processing module was spread over four nodes. In addition, this module was designed to re-use the memory for every band of operation, to reduce the memory requirement at any point of time.

The processed data from all the processing nodes was transferred to the controlling node, where it was packetized after applying required filtering and sent to other subsystem.[11]

Conclusion

The design of a computational intensive defense application should always consider a layered approach for modularity and scalability. Most of the applications have many requirements changes during the SDLC, which can be well accommodated only by a scalable architecture. The spreading of the application across multiple processors has to be carefully executed, considering the data transfer overheads, memory utilization and other important aspects.

References

1. Censier L M, Featrier P (1978) A New Solution to Coherence Problems in Multicache Systems,"
IEEE Transactions on Computers, 27(12):1112-1118
2. Gschwind M, Hofstee H P, Flachs B, Hopkin M, Watanabe Y, Yamazaki T (2006) Synergistic
Processing in Cell's Multicore Architecture. IEEE Micro 26(2):10-24
3. Intel Corporation (2010) Petascale to Exascale: Extending Intel's HPC Commitment. http://download.
intel.com/pressroom/archive/reference/ISC_2010_Skaugen_keynote.pdf. Accessed 11
January 2011
4. Sonics MemMax Scheduler. http://www.sonicsinc.com/uploads/pdfs/memmaxscheduler_
DS_021610.pdf. Accessed 10 January 2011
5. Mutlu O, Moscibroda T (2009) Parallelism-Aware Batch Scheduling: Enhancing both Performance
and Fairness of Shared DRAM Systems. IEEE Micro Special Issue 29(1):22-32
6. Ahn J H, Leverich J, Schreiber R S, Jouppi N P (2009) Multicore DIMM: an Energy Efficient
Memory Module with Independently Controlled DRAMs. Computer Architecture Letters
8(1): 5-8
7. The OpenMP Architecture Review Board (2008) The OpenMP Application Program Interface.
http://www.openmp.org/mp-documents/spec30.pdf.
Accessed 10 January 2011
8. Frigo M, Leiserson C E, Randall K H (1998) The implementation of the Cilk-5 Multithreaded
Language. Proceedings of the ACM SIGPLAN 1998 conference on Programming Language
Design and Implementation, 212-223
9. Culler D E, Gupta A, Singh J P (1998) Parallel Computer Architecture: A Hardware/Software
Approach. Morgan Kaufmann
10. Hennessy J L, Patterson D A (2006) Computer Architecture: A Quantitative Approach 4th
Edition, Morgan Kaufmann
11. Wikipedia article Hyper-threading. http://en.wikipedia.org/wiki/HyperThreading. Accessed
10.1.2010
12. Mars J, Williams D, Upton D, Ghosh S, Hazelwood K (2008) A Reactive Unobtrusive

Prefetcher for Multicore and Manycore Architecture. Proceedings of the Workshop on Software
and Hardware Challenges of Manycore Platforms 2008, 41-50
13. Nellans D, Sudan K, Balasubramonian R, Brunvand E (2010) Improving Server Performance
on Multi-Cores via Selective Off-loading of OS Functionalility. Proceedings of the

10th Workshop on Interaction between Operating Systems and Computer Architecture