Software Architectures for Real-time Embedded Applications for Broadcasting

Otto Železník[1], Zdeněk Havlice[2]

[1] Dept. of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic
o_zeleznik@yahoo.com
[2] Dept. of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic
zdenek.havlice@tuke.sk

Abstract. The paper discusses a choice of appropriate software architecture with regards to the specifications of embedded applications as information systems particularly used in area of radio and television broadcast audio program processing. The main requirement of mentioned embedded systems is an extremely good real-time response and minimized latency between input signal and output signal after processing. The requirements imposed to software architecture from the viewpoint of methods and algorithms which process the signal are in our case subjected to the minimal input-output latency requirement and specifics of hardware architecture and data structures used in embedded systems. The software architecture components and options are discussed and reviewed with their assets and limitations.

Keywords: software architecture, information system, artificial intelligence, real-time response, embedded application, broadcast processor

1 Introduction

We can think of the embedded systems in their particular area of focus as information systems since these basically analyze and process information in its specific form (signals, sounds, etc.). One of the information systems which belong to the mentioned group is an application of sound program processing in radio and television broadcasting. Even though the process itself, characterized by using proper psychoacoustic algorithms, has potential and function to improve-optimize the sound program for the audience, control of these algorithms is based on utilizing a simple fixed preset and coefficient methods. This approach is nowadays mostly used as a standard solution by the world top manufacturers. Another proposed method of control which is basically also a subject of my research and thesis is the analysis of sound program properties in relation to its informational content. This complex information is later processed in a carefully selected artificial intelligence method like trained neural network which should result in more optimal, natural and “intelligent”

process control algorithm e.g. resulting in improved sound from the human hearing viewpoint.

2 Broadcast embedded applications requirements

Today's broadcast embedded applications are designed exclusively in digital domain using modern DSP processors. The critical requirements in broadcast embedded applications are a minimal input-output signal latency and very good overall real-time response of the system. The first specification exists due to the nature of human hearing and sound perception [1], the second is a result of necessity to process signals with various feedback control signals. All of these requirements strongly affect choice of software architecture in the end. Recent research in the domain of embedded systems has demonstrated rather strong link between hardware and software architecture. Various limitations exist in software architecture choice due to the mentioned fact.

2.1 Memory size and software architecture choice in embedded systems

When considering for example a common PC platform where properties like memory size and/or CPU power are insignificant, these can accommodate very extensive and even complex software architectures and very large data structures with nearly no limitation.

Embedded systems on the other hand are mostly designed on smaller processors and less powerful hardware platforms where specific architectural factors are due. One of these factors is small available memory and specific memory model within existing CPU. This unfortunately strongly handicaps for example object oriented software architectures [7] due to their rather large memory requirements. They are almost impossible to implement with reasonable performance results.

Also memory model must strictly be questioned before the design is due since embedded hardware architectures [6] usually support only two access memory types of unequally partitioned sizes:

- Fast Layer 1 memories for execution critical code and data storage (available only in several kBytes)
- Slower Layer 2 memories for execution non-critical code and data storage (depending on CPU available up to max. few Mbytes)

There is also a cache support but with limited performance especially when executing critical code. Layer 1 memories are usually 3-10times faster than Layer 2 memories and full CPU performance is obtained only when executing code from Layer 1 memories. Adhering to the above facts, programmer or software architecture designed must carefully evaluate which data structure will be stored and operated from Layer 1 memory and which will be stored and operated from Layer 2 memory. The same applies for code separation as well.

Available memory size also affects the way how data and information is handled in the embedded system. Proper algorithm design helps reducing size of temporary data structures used for data processing. Using rather one common variable/buffer for data storage and processing in all processing algorithms is one way how to use memory properly. Moving less critical data buffers into Layer 2 memory and more critical data buffers into Layer 1 memory improves performance as well.

2.2 Programming language selection and software architecture choice in embedded systems

Choice or rather a necessity of using a certain programming language in embedded systems defines another group of constrains in software architecture selection possibilities.

Broadcast embedded systems require for some specific processing algorithms a very optimized and dense code since it may enhance execution performance. This is achievable only by designing hand-optimized assembly written routines and so avoiding use of any higher-level compiler available for embedded systems design. Programming and design experience indicates that even the best-of-the-class compilers are unable to achieve performance of well optimized hand-written assembly code. Gain in performance is about 5 to 10 per cent in favour of hand-optimized code. It is up to programmer to decide carefully which part of application requires such a high performance (routines running most of execution time) and which part of application can be designed using higher level programming languages like C and running least of execution time but with high software architecture complexity.

Broadcast embedded systems usually work fine with assembly language used for audio processing routines. These usually require very basic data structure types (circular buffers, simple variables) use with nearly no abstract models and only Layer 1 fast memory partitioning. On the other hand C language is mostly used for control algorithms which are of higher software complexity. Data structures become also more complex with use of dynamic memory allocation and management. They are however strictly located in Layer 2 memory region only. This complexity gain also gives a possibility to employ more complex and perhaps more useful software architecture components and interconnection in-between them.

2.3 Real-time response of embedded systems and software architecture choice

As mentioned earlier, one of the main broadcast embedded systems requirements is a minimal input-output signal latency and good overall real-time response.

The good overall real-time response is basically due in case of algorithms which are controlled from outside by external signals (feedback control). Since these need rather immediate reaction, they are strictly designed using hand-optimized assembly code and located in Layer 1 memory to assure safe critical execution.

Minimal input-output signal latency is another area of focus. In order to understand how this criteria affects software architecture choice, we need to know how signal is processed within the hardware architecture itself. One of the main principal requirements for correct digital signal processing is a sampling and processing theorem so-called Shannon-Kotelnikov theorem [2]. It generally requires that any input signal with a bandwidth F_{BW} is being sampled at sampling frequency at least 2* F_{BW} which ensures that the frequency information contained in the input signal is sampled without loss of any frequency component. Since human hearing is capable of sensing frequencies up to 20 kHz correct sampling frequency F_T would be 40 kHz. Practice over years of use by almost all professional world manufacturers indicates the standard to be 48 kHz [3] or 2^n multiplies for n = 0..2 [4]. Let the final F_T choice depend on the processing requirements.

If there was an ideal case with virtual hardware architecture, we could achieve the latency of the system as good as 1/ F_T. The only specification would be the ability of the hardware architecture to apply the process algorithm on every sample individually. Real situation is however rather complicated. First need in the digital signal process chain is to convert analog form of the sound into the digital representation and vice versa. AD and DA conversion serves for this purpose. Group of converters used for sampling audio signals due to its principle [2] unfortunately insert additional 2msec of signal latency into the digital stream. Tolerable overall signal latency based on human hearing properties is about 7msec [1].

Signal processing within the hardware architecture is basically controlled and managed using interrupt service routine. Each processing routine invocation requires several DSP service cycles (varying from 2 to 10 per cent of all available cycles on one sampling period, depending on DSP architecture). If invocation is done on every sample basis, the DSP spends service cycles pretty often, the overall input-output latency however would only be 1/ F_T however wasting a lot of computational power of DSP on mentioned service cycles. A solution to this computational loss is called a block processing technique [5]. Depending on maximum allowed input-output signal latency (can vary with application) a block processing technique is utilized using either short or long buffer sizes. Given signal processing algorithm (if a feasible implementation exists) is invoked only once per block of samples of size n which saves interrupt service routine cycles spent on every 1/n samples rather than on every sample processed. This improves overall performance of embedded system by 2 to 10 per cent. A general rule applies here, the bigger the block is the higher memory requirements are since larger memory block is allocated for samples buffer. Fortunately increasing block size is advantageous towards computational performance since the interrupt service routine is invoked less often and cycle saving becomes significant. A reasonable and often employed compromise for the block size is between 8 and 64 samples. This is however always individual to a given application. Thanks to the block-processing technique the DSP computational capacity as hardware architecture is 100 per cent loaded by the code execution and optimal performance is achieved while keeping the minimal input-output latency requirement due as well as achieving rather quick overall real-time response from the system. Employing block sizes from 8 to 64 samples the overall input-output latency values would be from 1 to 4msec depending on used sampling frequency. It is however necessary to mention that a latency gap created by difference between the critical

minimal latency (7msec) and the latency achieved by using block processing (1-4msec) is often consumed by signal processing algorithms usually of convolution or look-ahead type [2] which often employ rather bigger block sizes for samples stored temporarily for processing in the DSP memory.

The input-output latency requirement forces the embedded system software architecture to be designed so that it will be able to achieve such a performance. Most of the time, all of signal processing routines are designed using assembler hand-optimized code and executed from Layer 1 memory region for the highest performance. Only the simple software architectural components are allowed to use mostly in conjunction with circular buffers and direct variables.

3. Software architecture components for broadcast embedded systems

So far we have been speaking rather of broadcast embedded system implementation and technical requirements. While it is crucial to adhere to these specifications strictly, they result in rather limited options for choosing broadcast embedded system appropriate software architecture. There is no general software architecture available fulfilling all mentioned needs. Embedded systems however typically involve a combination of one or more partial software architecture types which in proper utilization and combination create a very well performing software system. Basic partial software architecture types [7] which adhere to these specifications are the following:

- *Domain-specific systems:* could be described as "reference" systems for a domain specific area of application. By specializing the architecture to the domain it is often possible to create an improved the descriptive power of structures used in the architecture. Embedded systems software architecture is assumed to be a domain-specific system indeed.
- *Process control components*: these architectural components are basically intended to provide very dynamic control over a signal processing environment with rather instant real-time response. In embedded systems, these are assembler written hand-optimized codes with very limited data structures focusing strictly on performance. They usually work with instant feedback control signals.
- *Pipes and filters:* are based on a structure of a black-box in the middle with a set of inputs and outputs. The black-box is the processing algorithm itself whereas it processes the inputs and generates the outputs. Their advantageous property is they could be interconnected in between each other in various orders thus creating possibility of highly configurable system if processing order on signal does make a difference in the final result. One of the example implementation in the broadcast embedded system is any general audio processing algorithm used on a common circular buffer of audio samples. The advantage of using pipes-and-filters architecture here is that the order of

process is easily defined since all algorithms work on the same data structure here. This gives programmer and user an easy way to modify the process itself on-the-fly in dynamic fashion thus achieving different process behaviors. The pipes-and-filters architecture is very well implemented in higher-level languages such as C due to its descriptive power to abstract structures rather than in assembly written code which is more difficult to approach from the above mentioned point of view.

- *Layered systems:* are ones of the most used software architecture components in embedded systems. Since a very precise scheduling of tasks is required in such systems, these are most useful for the given purpose. Some of the layered systems may even involve hiding certain outer layers from inner layers thus creating a sort of "protected" environment for sharper overall stability of the whole system. A good example is a real-time OS driven embedded application where core and peripheral drivers are the most inner layer in the system and the user applications are the most outer layer, communicating with the core via communication and scheduler layer. Inner layers usually also serve as interrupt service routines for sampling and processing. Since scheduling reliability of these must be 100 per cent they are usually (with the OS core when implemented) on the same layer with highest priority of execution. Otherwise missing process for input sample causes strong and audible signal distortion and degrades system performance significantly.
- *Event-based invocation:* is a specific software architecture component which is useful for controlling user input and implementing user interface. The advantage of this architecture component is in how algorithms can be layered by their priority and still be able to communicate-invocate themselves in event-based fashion with very good real-time response. One good example for system of this kind is a simple volume regulator. The sound processing part of algorithm is running at very high priority applying gain constant which is read from the control component volume, while there is another software component on the outer layer priority which does not need such sharp timing and can run much slower (difference in schedule intervals is more than 1000 times). The algorithm there serves reading the control component volume and storing appropriate gain value into the common gain variable used by both components. Another useful example is generating events on button components when these are pressed-released. Each given button has assigned its own software component which is always invoked only when an event occurs on the button itself. More event-based invocation schemes are available and often used in the DSP embedded applications not only for broadcasting.

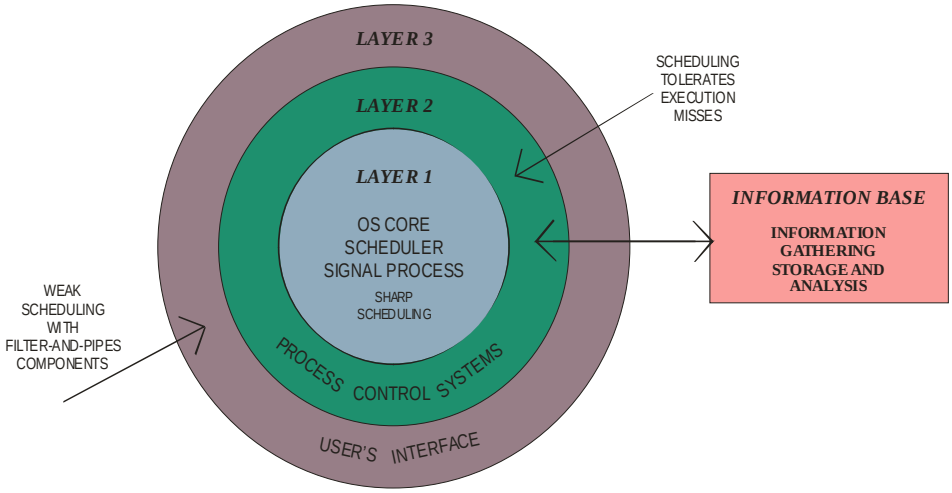

Figure 1: Broadcast embedded system software architecture model

As we have already mentioned the necessity of a proper combination of partial software architectures to achieve well performing overall system, over several years of extensive development in area of broadcast embedded applications I always ended up in the basic domain-specific software architecture as depicted in Figure 1.

It is performance oriented and strictly adheres to the requirement of a minimal input-output latency still keeping user-programmer a possibility of generating quite dynamic and easily reprogrammable and customizable system.

As a basis here is used distributed layered system with several components including architectural elements of other types. Core scheduler for user's interface components is on the Layer 1 in conjunction with all signal based processing for its strict schedule timing requirement. The signal processing software components are usually written in assembly language due to weak code optimization performance of higher-level languages that however depends on hardware architecture used. The layer above in the model is usually serving for components which still need relatively sharp scheduling but missing few scheduled executions will not degrade system performance. Here are included analysis tools and routines which gather information from the signal and communicate with information base which is another part of distributed system. The outermost layer is usually implementing user's interface written strictly in higher-level languages like C. Good combination at this layer is a use of event-based invocation components as well as auxiliary routines in filter-and-pipes fashion. This software architecture model gives a very good overall performance and helps use DSP computational capability up to 100 per cent at nearly full execution time.

Embedded system depicted as in Figure 1. is considered an information system because it also serves as analysis and storage function for incoming information/data represented by signal itself. The signal is analyzed by appropriate signal processing techniques and by artificial intelligence analysis algorithms resulting in a stream of knowledge information (musical genre, program type, voice genre, history of styles changes, commercial breaks history) stored in the distributed information base where

it is later used for several statistical analysis and direct purposes for optimizing neural network for signal process control. Information base is rather large due to the fact that sound as a signal after digitalization represents lots of data necessary to analyze resulting in rather large information data as well. This is why the choice is rather in favour of distributed system.

4. Conclusions

Continued modernization and improvement of the hardware DSP architectures enables implementing and using more and more complex software architectures in various areas of applications. Even more parallel functioning DSP processors challenge software optimization to higher levels and using improved parallelism they achieve sustained performance gain. This paper discusses aspects of appropriate software architecture selection for broadcast embedded applications used in broadcast processors. It faces strict philosophy requirements such as minimal input-output latency and analyses how this specification compromises options for software architecture selection. Paper also proposes a performance oriented software architecture model for broadcast oriented embedded applications including distributed information system components for data processing and storage. Further study is possible in area of improving software architecture to achieve better portability and re-use using higher-level programming languages even in inner layers of the architectural model, perhaps optimizing model for information base within the embedded system itself.

This work is supported by VEGA 1/2176/05 – Technologies for Agent-based and Component-based Distributed Systems Lifecycle Support.

References

[1] Gold, B., Morgan N.: Speech and Audio Signal Processing: Processing and Perception of Speech and Music, Hardcover, August 1999
[2] Lathi, B. P.: Signal Processing & Linear Systems. Oxford University Press, New York (1998)
[3] Mapes-Riordan, D.: A Worst-Case Analysis for Analog-Quality (Alias-Free) Digital Dynamics Processing, 105th Convention of the Audio Engineering Society (AES), USA San Francisco 1998
[4] Foti, F.: Digital Dynamic Processing: “It’s All In The Samples!”, A Omnia Telos Company White Paper, Cleveland OH USA 2002
[5] Ko, M., Shen, Ch., Bhattacharaya, S..: Memory-constrained Block Processing Optimization for Synthesis of DSP Software, International Conference on Embedded Computer Systems, Samos Greece 2006
[6] Analog Devices.: ADSP-BF533 Blackfin Processor Hardware Reference, Analog Devices One Technology Way, USA 2003
[7] Garlan, D., Shaw, M..: An Introduction to Software Architecture, CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21