A STUDY ON EMBEDDED SYSTEM

Neelam Rani[1], Yamini Mathur[2]

UG, [1,2] Department of Electronics and Communication Engineering,
Raj Kumar Goel institute of technology for Women, UP (India)

ABSTRACT

Many embedded system have substantially different design constraints tan desktop computing Applications. No single characterization applies to the diverse spectrum of embedded systems. However, some combination of cost pressure, long life-cycle, real time requirements, reliability requirements, and design culture dysfunction can make it difficult to be successful applying traditional computer design methodologies and tools to embedded applications. Embedded system in many case must be optimized for life-cycle and business-driven factors rather than for maximum computing throughput. There is currently little tool support for expanding embedded computer design to the scope of holistic embedded system design. However, knowing the strengths and weaknesses of current approaches can set expectations appropriately, identify risk area to tool adopters and suggest ways in which tool builders can meet industrial needs.

Keywords: Real time operating system (RTOS), Read only memory (ROM), Random access memory (RAM), Input Output (I/O), Analog to Digital (A/D), System on a chip (SOC), *M***oving** *P***icture** *E***xperts** *G***roup** ***(MPEG), Automatic Teller Machine (ATM), Application Specific Integrated circuit (ASIC), Matrix laboratory (MATLAB), Digital signal processor (DSP).***

I. INTRODUCTION

If we look around us, today we see numerous appliances which we use daily, be it our refrigerator, the microwave Oven, cars etc. Most appliances today are powered by something beneath the sheath that makes them do what they do. These are tiny microprocessors, which respond to various keystrokes or inputs. These tiny microprocessors, working on basic assembly languages, are the heart of the appliances. We call them embedded systems.

II. HISTORY

The first recognizably modern embedded system was the Apollo guidance computer, developed by Charles stark draper at the MIT Instrumentation Laboratory. At the project's inception, the Apollo guidance computer was considered the riskiest item in the Apollo project. The use of the then monolithic integrated circuits, to reduce the size and weight, increased this risk.

IJARSE

The first mass-produced embedded system was the guidance computer for the Minuteman missile in 1961. It was the Automatics D-17 guidance computer, built using discrete transistor logic and a hard disk for main memory. when the Minuteman II went into production in 1966 the D-17 was replaced with a new computer that used integrated circuit, and was the first volume user of them.

III. EMBEDDED APPLICATION DEVELOPMENT CHARACTERISTICS

What characterizes an embedded system? Usually it means that there are a set of predefined, specific functions to be performed, and that the resources available (e.g., memory, power, processor speed, computational functionality) are constrained. Often, though not always, the application will run out of ROM on a microprocessor. This, in comparison to a desktop computer, which is a general-purpose processor and support system designed for a wide range of applications. The range of embedded software is much broader than desktop software, where a handful of applications (word processors, spreadsheets, games, and so on) makes up the vast majority of applications.

The most developed segment of the embedded tools market are the off-the-shelf real-time operating systems (RTOSs), including their support programming tools: source level debuggers, integrated development environment, and compilers.

IV. COMPONENTS NEEDED

The components needed for the development of Embedded Applications are:

Micro controller.

Real-time Operating System.

A language for coding.

Machine code generator.

Debugger.

4.1 Micro Controller

The micro controller consists of microprocessor, ROM, RAM and some I/O ports all on the same chip. The commonly used micro controller is 8051.

4.2 Machine Code Generator

The source code is written in particular language like C/C++. This has to be then used to generate the code for the particular microprocessor. The machine specific code is then burnt into the chip(ROM). The microprocessor takes the instructions from the ROM and executes them to produce the desired effect.

4.3 Real-Time Operating System

What does “real-time” mean when used in the context of an operating system? Simply put, this means that the embedded application can and will respond to external events in real time, as opposed to waiting for some other task or process to complete its operation. This is made even more confusing by the use of the terms “hard real-time” and

IJARSE

"soft real-time."

Hard real-time means an activity must be completed always by a specified deadline (which may be a particular time or time intervals, or at the arrival of some event), usually in tens of microseconds to few milliseconds. Some examples include the processing of a video stream, the firing of spark plugs in an automobile engine, or the processing of echoes in a Doppler radar.

Soft real-time applies to those systems that are not hard real-time, but some sort of timeliness is implied. That is, missing the deadline will not compromise the system's integrity, but will have a deleterious effect.

V. DESIGN ISSUES FOR EMBEDDED SYSTEMS

Embedded System are, if nothing else, characterized by constraints such as response, size, performance, cost, and so on. Numerous questions have to be answered before the design even begins:

1. What are the worst case performance requirements for each activity?
2. What are the number and complexity of activities to be performed?
3. What is the degree of coupling of tasks?
4. How much RAM and ROM will be consumed for specific set of tasks?
5. How much RAM and ROM does the hardware design provide?

A number of more commonly faced issues are summarized here.

5.1 Time Constraints

Real –time operating system bow to a combination of time specific constraints. some routines must execute at precisely fixed intervals, while other routines are not bound to critical time alignment. The most critical task of an embedded programmer is to characterize each of the actions to be performed so he will know how to assign priority and resources to that action in order that the overall system performance objectives are met. To aid in this task, it is helpful to break the actions in an embedded system down into the following four task groups:

1. Time critical task routines are those that must occur at a fixed rate with a minimum startup latency (e. g., servicing an A/D converter).
2. Time sensitive task routines are different from time critical tasks in that they can tolerate a large latency before being serviced. Like time critical task routines, they may also occur at fixed rates or they may be initiated at random intervals, but are guaranteed to execute no more frequently than some fixed rate by the task handler itself.
3. Idle task routines are important background operations, and they execute as frequently as possible at more or less random interval when it is convenient.

IJARSE

4. Mainline tasks routines interpret the user commands, perform non-real-time functions, and make calls to the time sensitive and idle task service routines.

5.2 Safety

Probably the first and simplest techniques learned by many embedded programmers consists of filling unused program memory with either halt instructions or illegal instructions. This techniques guards against illegal jumps outside of the program space and provides cheap insurance.

Another common protection is to use buffers that guard against stack underflow/overflow or the corruption of a task's stack. Many of the commercial RTOSs now contain facilities and functions that support stack checking.

To verify the integrity of a program or data stored in ROM, a simple ROM test should be included as well a watchdog timer to prevent the software from getting caught in a loop.

5.3 Device Drivers

It is well known that writing efficient device drivers requires knowledge of both hardware and software. The resulting device drivers become the keys to embedded system performance since they are called repeatedly, and therefore dictate real-time performance in terms of response time, and utilization of memory and other system resources.

5.4 Storage Allocation

One important feature to be considered in the selection of an embedded system design is storage allocation. Ill designed dynamic storage allocation can be wasteful for two reasons. First, allocating memory from the heap can be both slow and non-deterministic. The time takes for the manager to search the free-list for a block of the right size may not be bounded. Second, one may create the possibility of a memory allocation fault caused by a fragmented heap. One typical solution is to statically declare all objects up front always exist ant take space. Whilst difficult, the apparently conflicting goals of a dynamic storage allocator can be achieved.

5.5 Optimizing Performance

Writing embedded code that runs efficiently brings about a whole new set of rudderless. Often optimizing for speed and size opposing design goals-an improvement on one often degrades the other. In trying to achieve this balance, the article promotes the use of three techniques:

1. The judicious use of the optimization option found with most embedded cross-platform compilers (for example, eliminating redundant code, or replacing operations with equivalent but faster operations, or unrolling loops optimizing the use of registers, or removing code segments that the compiler knows cannot be reached)
2. The mix of fixed and floating-point operations.
3. The employment of users optimizations, making the most out of available resources.

IJARSE

5.6 Debugging memory Problems

Since many RTOs and/or embedded microprocessors do not support memory protection, tracking down software memory bugs can become a serious debugging problem. In attacking this problem, it is best to categorize the problem by the type of Memory affected. In tree general , they fall into three categories:

1. Global memory bugs: those bugs that result in corruption of global memory data areas.
2. Stack memory bugs: these often cause a complete failure of the program execution; they are the hardest to track down as they are often a function of external events and the current state of the stack.
3. Dynamic allocated memory bugs: examples are, heap memory allocated by a mallow service; or problems caused by writing past the boundaries of an allocated memory block or using one that is no longer allocated.

VI. FUTURE TRENDS IN EMBEDDED SYSTEM RTOS

There is a flood of trends rushing through the embedded market today, many influencing the RTOS requirements in conflicting ways. It is hard to envision that five years from now RTOS products will bear much resemblance to what is supplied today. Some of these trends are application driven while others are device driven, and it is important to understand the influences these trends will have.

6.1 Application Specific

In several markets, the end users have banded together to issue specific requirements for RTOSs to be used in their future products. They have purposely chosen to drop their proprietary behaviors of the past in order to get the benefits of multiple suppliers and interoperability of software. In this manner, only the needed software is linked into the application, preventing additional overhead and allowing for an extremely efficient kernel implementation.

6.2 System On A Chip (Soc)

As mentioned earlier, SOCs are beginning to appear throughout the embedded markets, in al least three different ways. First, the semiconductor suppliers are providing developers the ability to pick and choose from a combination of industry standard functions integrated around a 32- bit core processor. These functions may include memory, IO drivers, a bus interface, network protocol support, or algorithms for special functions, such as an MPEG decoder. Second, end product manufacturers are integrating custom ASICs with common 32-bit core processors to provide complete solutions. Some recent examples include cable modems and ATM switches. And third, startups are emerging that will provide custom design services, complete with optimized RTOS, compiler, and debuggers.

SOC will be particularly well suited for a whole range of consumer electronics and wireless communications devices where size, cost, and low power requirements are crucial. It will also drive cost reductions in networking and telecom equipment, where more functionality can be added at lower costs. A subset of this SOC trend is the

IJARSE

emergence of multi-core devices on single silicon. The most common to date has been the combination of standard microprocessors and Digital Signal Processors (DSPs). In some cases, the DSPs are dedicated functions processors, but emerging trends have the DSP as a full programmable device.

6.3 Automatic Code Generation

Probably the most radical notion is the idea that application code can be generated automatically from graphical diagrams depicting the logic flow for the end product. To a limited extent, this has already been accomplished, with product like MATRIX, MATLAB being used for application modeling and code generation. In the case of MATRIX, flight ready code for the international space station has been used for some time now, and the technology is being extended into the more restrictive automotive market. If these tools were to become reality, the whole nation of commercial RTOS and development tools will be upset, as the developer will only interact with the graphical tool, and will be totally isolated from the resulting software implementation.

VII. CONCLUSION

This paper helped in understanding the following concepts.

1. Embedded systems application development.
2. The components of embedded systems.
3. The design issues.
4. The applications in which embedded systems are used.
5. RTOS and its features.
6. How to carry out effective presentation.

REFERENCES

[1] Bernard Cole, "*Architectures overlap applications*", Electronic Engineering Times, March 20, pp. 40,64-65, 1995.

[2] Stephanie White, Mack Alford & Julian Hotlzman, "*Systems Engineering of Computer-Based Systems*." In: Lawson (ed.),Proceedings 1994 Tutorial and Workshop on Systems Engineering of Computer-Based Systems, IEEE Computer Society, Los Alamitos CA, pp. 18-29, 1994.

[3] "*Design Automation for Embedded Systems*": an international journal, Kluwer Academic, ISSN 0929-5585.

[4] "*Embedded Systems Programming*", Miller Freeman, San Fran-cisco, ISSN 1040-3272.

[5] Daniel D. Gajski, Frank Vahid, Sanjiv Narayan & Jie Gong, "*Specification and Design of Embedded Systems*", PTR Pren-tice Hall, Englewood Cliffs NJ, 1994.

[6] Jack Ganssle, "*Art of programming Embedded Systems*", Aca-demic Press, San Diego, 1992. Don Thomas & Rolf Ernst (eds.),Proceedings: Fourth Inter-national Workshop on Hardware/Software Co-Design, IEEE Computer Society, Los Alamitos CA, 1996.

IJARSE

[7] David Patterson & John Hennessy, "*Computer Architecture: a Quantitative Approach*", Morgan Kaufmann, San Mateo CA,1990.

[8] Philip Koopman, "*Perils of the PC Cache*", Embedded Systems Programming, May 1993,6(5) 26-34.

[9] Shem-Tov Levi & Ashok Agrawala, "*Fault Tolerant System Design"*, McGraw-Hill, New York, 1994.

[10] Daniel Siewiorek & Robert Swarz, "*Reliable Computer Systems: design and evaluation"* (2nd edition), Digital Press, Burlington MA, 1992.

[12] Nancy Leveson, Safeware: system safety and computers, Addison-Wesley, Reading MA, 1994.

[13] Georgette Demeset al., "The Engineering Design Research Center of Carnegie Mellon University,"Proceedings of the IEEE,81(1) 10-24, January 1993.

IJARSE