RoweBots
Research Inc.

White Paper – 22 Sept 2009

Microcontroller Lean Product Development

Author: Kim Rowe,
CTO, RoweBots Research Inc.

Unison is a trademark of Rowebots Research Inc. All other trademarks are the property of their respective owners

Introduction

Microcontroller based products are becoming much more complex and have now evolved to System on Chip (SoC) components which provide everything on a single chip. This has created a need to reconsider the software approaches used to ensure that systems are well designed using these new parts.

Silicon vendors of SoC MCUs are now delivering software modules that can be used to test their hardware or implement I/O features. It is a great place to start as these modules typically come with demo programs which can be used to ensure that certain features meet specific design goals. The main issues with these components are: quality control and testing, standardization, licensing limited to a specific vendor's MCUs and an assumption of a single loop of control.

The main issue the designer must face is not the design of the hardware, it is the software design. The hardware is totally fixed and now the software must support this specific MCU hardware.

How are users to effectively build OEM products largely tailoring software and levering these SoC MCUs and at the same time maximize profits, minimize risk and minimize time to market? Some might think that this is not the correct question, that analysis should focus on technical details of various software approaches and components. By comparing all solutions based on their overall long term cost, the OEM company will achieve their corporate goals in the R&D, manufacturing, marketing and financial areas.

Requirements for OEM Solutions Are Shifting

Most OEMs operate in a very complex space. Competitive pressures are high which mean that time to market requirements are stringent and features are dated more quickly. Shifting demand for energy efficiency and environmentally friendly products is shifting features sets more quickly requiring completely new designs. Also new networked features are expected, color touch panel displays are required to convey quality and sell product, and advanced digital accelerometers and other advanced sensors create a very dynamic scenario for developers

In this rapidly changing and completely unknown environment, how are designers able to develop excellent solutions that also have requirements for:

- field diagnostics and testing,
- fast, simple and accurate manufacture,
- constructed easily, inexpensively and quickly,
- software based architectures and features,
- centralized and distributed architectures,
- full product line support,
- and simple maintenance.

This is just the beginning of a typical requirements list, generally many other desirable features are added.

What are the success criteria for a design? This is also key when considering a software architectural approach. The design must:

1. Function well in practice.
2. Minimize BOM cost.
3. Minimize manufacturing costs.
4. Be easy to maintain.
5. Maximize reliability.
6. Meet all specifications.
7. Support lean product development.
   - **a.** platform based.
   - **b.** open standards based.
   - **c.** portable OS and applications.
   - **d.** incremental expansion for new features.
8. And ensure Minimal Life Cycle costs.

>

Rowebots Research Inc. © 2009,

Software Implementation Options

Assume that the architecture is a network of MCUs in the simplest case only a single MCU and supporting hardware components. The software architecture is one of four types with one extra optional property – the defacto industry standards for embedded systems today, POSIX and Linux™.

1. A single loop of control (no standards here)
2. A simple real time kernel (POSIX/Linux or proprietary)
3. A real time kernel with integrated I/O (POSIX/Linux or proprietary)
4. A real time or embedded operating system (POSIX/Linux or proprietary)

The overall summary of alternatives is presented in *figure 1*.

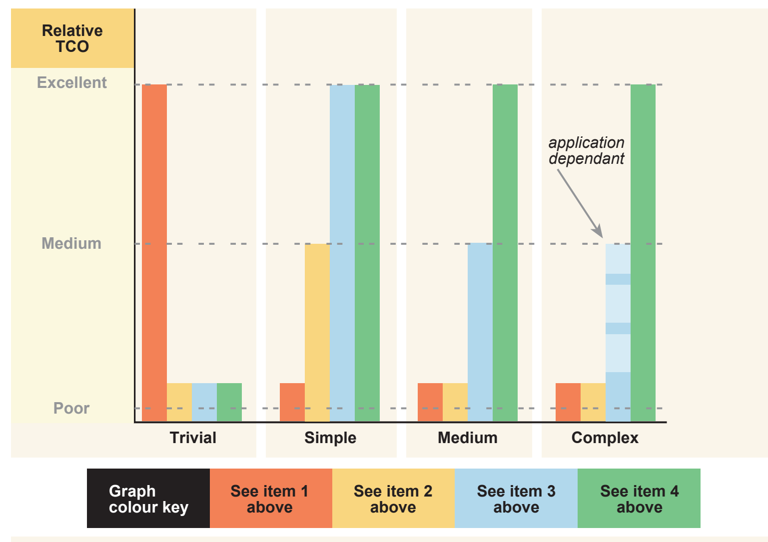

Figure 1: Total cost of Ownership by Software Architecture

Analysis of Software Options

Single Loop of Control

In this case, there is neither a proprietary or a standards based solution – there is just the designers custom approach. There is no platform based approach and all the lean development benefits are lost as well as ease of maintenance and maximum reliability since the system will be undocumented and almost untested. Incremental enhancement is severely limited and applications often break during expansion.

Here the total life cycle costs have been maximized across the product line. All maintenance and future integration cost is undertaken by the user, The scheduler is essentially hand coded in the application and is brittle. Each product must be individually maintained – an expensive proposition that many MCU developers are not avoiding today.

This choice represents the worst of all worlds unless your application is trivial.

Real-Time Kernel Plus I/O Libraries

The real time kernel approach can be efficient for small systems with completely non standard I/O; however, this is seldom the case. Because there is no I/O or I/O model included, if this is required the user must invent it, document it and test it. If the kernel is insufficient, the user must reinvent it, document it and test the kernel. All this takes a great deal of time and money.

Analysis of Software Options

continued

Real-Time Kernel Plus I/O Libraries

Some developers choose to purchase a kernel and then integrate their own I/O modules or other public domain or other vendor's modules. This overall approach suffers from the fact that the user must integrate all the different modules (which involves modification to the core software) and then maintain them. This is very inefficient because the user ends up supporting the entire environment including all of the integrated I/O modules

Some vendors hang their entire set of benefits on the fact that they do one small feature better than others. This type of marketing often engages technical people who are looking for the perfect technical solution; however, they missed the forest because they concentrated on a single tree. The real cost issues are not overcome using this approach.

The unfortunate part of using a kernel with vendor provided I/O is that the libraries are not free. Although they don't cost anything, and are free from the vendor, they represent lock in to a specific vendor. Now your application is fixed to a specific family of MCUs which could easily fail to meet future needs. In addition, you must still port the code to your selected kernel, define an I/O model, implement and test the I/O model and then, integrate and test the system. Of course all maintenance falls on your shoulders too.

Another approach involves replacing these vendor libraries with open source (non GPL) libraries. This approach is portable if the kernel is portable and there is no vendor lock in – all good news. The bad news is that the open source code now needs full porting, integration and testing along with the definition of an I/O model, implementation and testing of the I/O model. Maintenance is also required.

What is the cost of this approach? The costs are:

- Lost time to market across the entire product line
- High integration and testing costs including test suite development
- Kernel cost (if purchased)
- Documentation costs
- Training costs associated with the proprietary solution
- Maintenance costs for this now custom software implementation

The cost of this work and the lost market share dwarf the cost of a purchased RTOS in all but very rare cases.

Proprietary RTOS

Major cost reduction can be achieved using an off the shelf proprietary RTOS. It can eliminate time to market, reduce risk and minimize maintenance and documentation costs. The biggest issue with these solutions are they are proprietary and you have again locked yourself in to a specific vendor.

Analysis of Software Options continued

Open Standards Based RTOS

Developers can achieve all the benefits of a proprietary RTOS and much more by selecting an open standards based RTOS. In fact, this is the reason that Linux and POSIX are the standards for all RTOS or embedded operating systems today. Linux and POSIX are the defacto choices of most developers. The architeture for Linux (and µCLinux) is shown in *figures 2a-2e* (compare the complexity to Unison™ and DSPnano™ in *figure 3*).

Why can't you just run Linux on your SoC MCU? This seems like the ideal solution. You get:

- open source
- millions of lines of code to reuse (the easiest means to reduce your costs)
- a broad set of I/O modules
- support for many processors, and more.

The real catch here is that Linux is too large and not nearly modular enough to run on a tiny SoC MCU. In addition, it uses GPL, which makes all of its source code useless for embedded OEM designers which don't have an MMU.

Why can't you run µCLinux? It supports an MMUless processor. Shouldn't it be ideal?

- The fact is that it still carries with it many of the problems of GPL.
- It is also too large by a significant amount, it simply won't run on SoC MCUs.
- It has all the modularity and configuration issues of Linux, and
- it is non standard in its calls for basic functions.

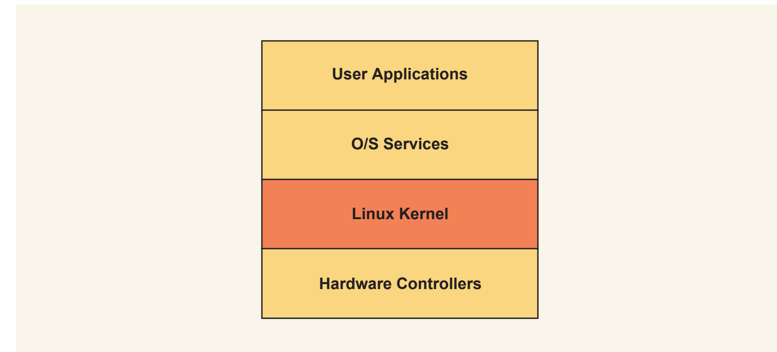

Figure 2a: Kernel Overview

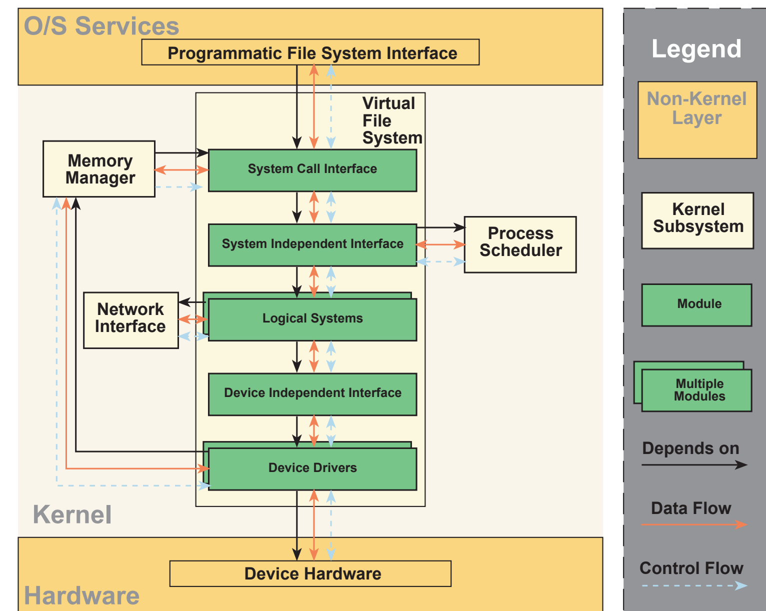

Figure 2b: Virtual File System

Open Standards Based RTOS

What are the options for an open source, non GPL, ultra tiny embedded Linux and POSIX compatible RTOS? Today there is really only two solutions that run in this environment – DSPnano RTOS and the Unison RTOS. Unison offers ultra tiny embedded Linux and POSIX compatible 32 bit processor RTOS features and DSPnano offers 8/16 bit processor RTOS features. The Unison architecture is shown in *figure 3* (compare the complexity to Unison and DSPnano in *figure 2 a-e*). Both DSPnano and Unison come out of the box with 20+ demos and complete I/O solutions so you are up and running in 10 minutes.

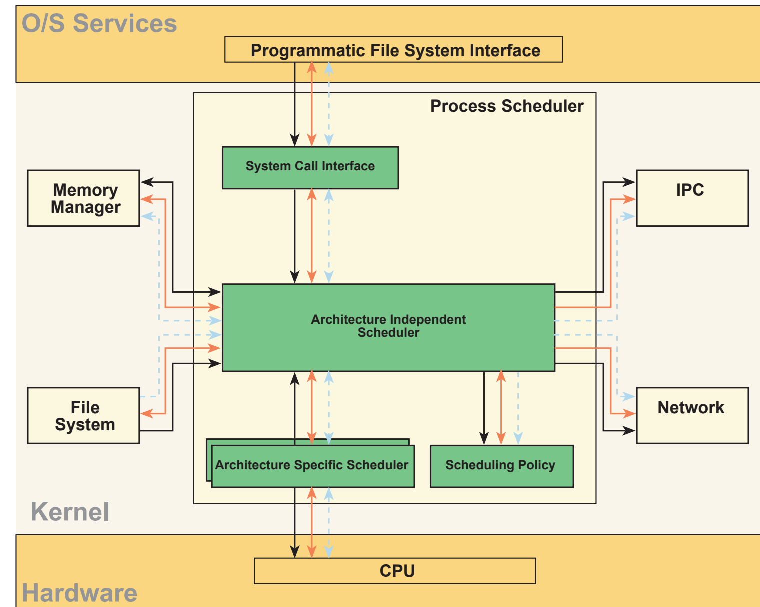

Figure 2d: Process Scheduler Details (see figure 2c for Legend key)

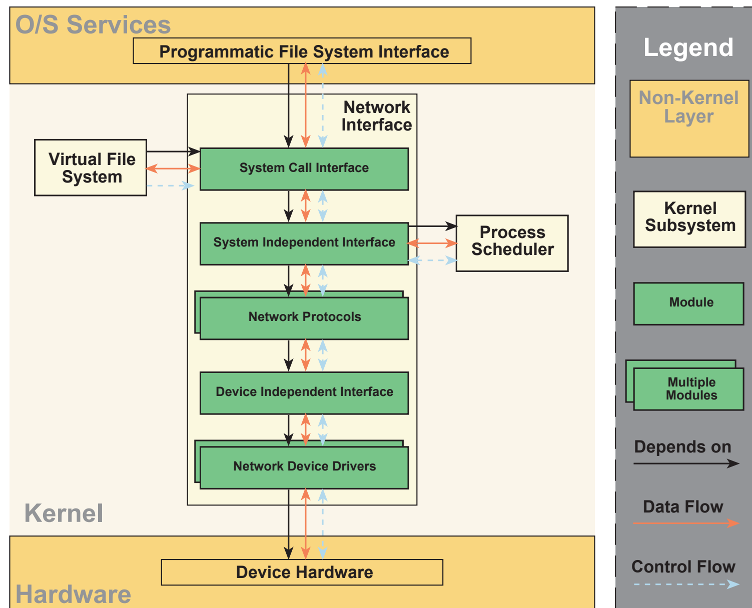

Figure 2c: Network Interface Subsystem

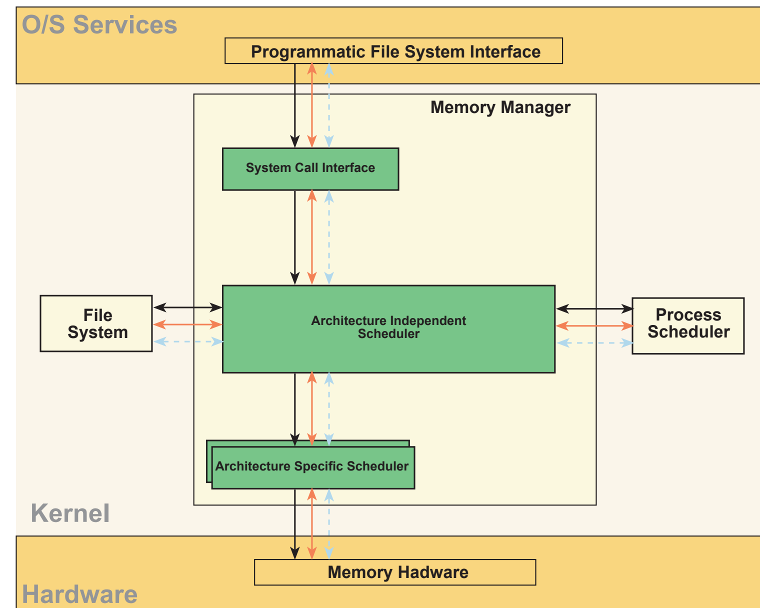

Figure 2e: Memory Manager Details (see figure 2c for Legend key)

Rowebots Research Inc. © 2009,

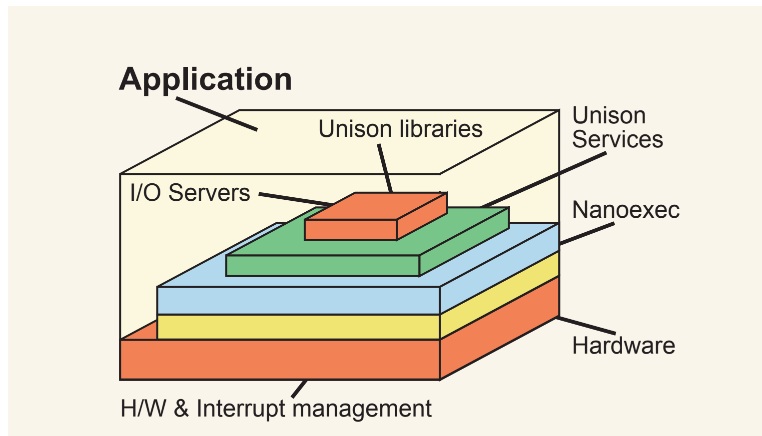

Figure 3: UNison Architecture

Summary

Using a single loop of control, a real-time kernel with vendor I/O modules, a real-time kernel with open source I/O modules or a proprietary RTOS are inferior choices when maximizing profits and minimizing time and risk for embedded OEMs. By selecting open standards based RTOS products for your SoC MCU based products you eliminate time to market, eliminate integration, get trained people easily, have clear communication in your team, off the shelf I/O and more.

By selecting Unison RTOS or DSPnano RTOS as you open standards based RTOS you can be running out of the box in 10 minutes with a broad set of I/O on a broad set of SoC MCUs – all with open standards.

White Paper – 22 Sept 2009

Microcontroller Lean

Product Development

Author: Kim Rowe,
CTO, RoweBots Research Inc.

Contact Information:

Kim Rowe, Founder
sales@rowebots.com
+1 519 208 0189
+1 519 498 6917

RoweBots
Research Inc.

Rowebots Research Inc. © 2009,