EMBEDDED SOFTWARE ARCHITECTURE

By Andreu Sabé,
Software architect SALICRU

salicru

Introduction

For long time it has been considered that, a key requirement to reduce time-to-market for SoC (System-On-Chip) devices is to start software development early in the design process. Since this approach has already been established, the next challenge is to mature Embedded Software engineering. This includes Embedded System Software Architecture and its specification.

According to various reports, an embedded project takes an average of 13.1 months to complete if started in a classical way, but this time can be drastically reduced by using an Embedded System Software Architecture.

Empower your competitiveness for Faster Time-to-Market and System reliability

This technique is becoming a key issue at the time to reduce the Time-To-Market and have a better reliability. At the same time, the Embedded System Software Architecture will satisfy the product requirements and be within the memory and speed capabilities, among others, of the corporative processors.

Applying software architecture for embedded reuse is an area identified as not being fully explored in current company developments.

Below, there are several topics to be enhanced with the help of software architectures:

- Easy-to-use application libraries.
- Easy-to-reuse application libraries.
- Proven and bug free common parts.
- Easy-to-port to different hardware systems.
- Easy-to-test as it has Hardware Abstraction.
- Multiple-core based platforms to run on.

ESSA introduction

An Embedded System Software Architecture (ESSA from now on) will be a mid-term benefit for any company that develops their own products based on microprocessors/processors. This covers a huge network of companies along with their own R&D departments.

Typically, when a new project "kicks off", the first consideration to have in mind is to reuse the software and hardware from previous developments and start from that point on. This is how it is done, re-adapting the source code and hardware design.

This approach has some disadvantages as follows:

- The re-usability degree could be higher.
- The reused source code is not mature enough since it has been adapted (not used as it was).
- The resulting source code is not optimal.
- The resulting source code cannot be reused for the next development generation.
- The source code quality is downgraded.

ESSA Basics

A variety of components will be taken into account to reflect the system requirements at not only development time, also production processes and After Sales service will be covered because the aim of an ESSA is to have a complete WIN-WIN solution.

The first issue to be thought of is what exactly should be done and which the resources in terms of Core capabilities and System requirements are. Real time, Memory Usage, Data Types, Storage, System Interrupts, Timers, Shell...are key topics to be considered.

ESSA Benefits

There are some Ready-to-Use Commercial ESSA that can be used, some of them even come with hardware, documentation and examples. This could be an easy way to start "playing" with an ESSA and make the first approach.

Another way could be to start developing an own or proprietary ESSA. This would be a mid-term investment and require some technical resources for some time.

Anyway, in the end, both solutions would end up with SoC developments based on an ESSA and start enjoying its benefits.

If the company has a limited R&D team, the first approach is the one recommended but as a mid-long-term solution, the second approach has a better fulfilment to the product development and also eases the product Manufacturing and After Sales Service processes, which is in the end the major benefit of an ESSA.

ESSA Parts

These are what we consider as key parts for the whole process coverage:

- Minimal Operative System with Task dispatching and Timer control.
- Hardware Abstraction Layer.
- System Shell.
- PC based ESSA compatible Application.

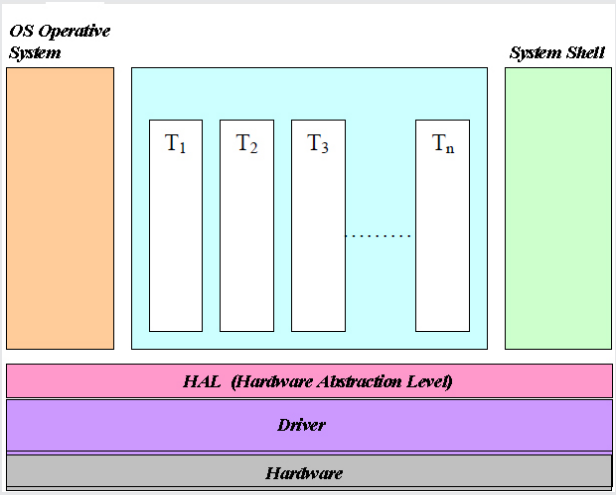

Figure 1: Example of ESSA block diagram

Basics of ESSA

Operative System

Along with the System Shell and HAL (refer to next sections), the Operative System (OS from now on) is being considered as a pillar where the ESSA grows. Its main purpose is to dispatch the Tasks and give the System the required Time determinism. At the same time, it controls the start/stop/standby/resume processes defined by the system properties.

Some OS also give the system the multi-task capability, so in that sense it is important to determine if this is a feature to be considered, otherwise the ESSA can be oversized or underused depending on the point of view

Another capability the OS will provide the System with is a series of common resources such as Time control, Event Time Control, File System, Access Control, RTC. From any Task of the software Application, a resource can be requested to the OS and released later when is no longer needed so another Task can use it.

A key issue is that, if all the stuff above is concentrated on the OS part, any application or future developments can reuse these features entirely without the need of implementing them again, or even teaching new engineers an alternative way to dispatch or control the time determinism because the software stack is different

On the other hand, the OS is only linked to the Hardware at the time to request the time base, which would probably be an OSApi to link the OSlayer to the processor hardware. This would ease the portability of the software from one hardware platform to a different one.

RAM Memory Usage

One important issue to be considered in an ESSA is the use of RAM memory. A part from the amount of RAM memory the different Tasks need to fulfil their requirements, the ESSA has also its own memory needs. The ESSA is intrinsically layer based so it uses more Stack memory compared to “classic” software applications and then, depending on the type of OS, even more RAM memory is needed. This can be seen as an important drawback but in our opinion, if the right type of OS is chosen, the benefits will compensate this inconvenient.

Another factor that helps to mitigate the importance of this drawback is the fact that, day after day, new processors with a better price/RAM ratio are released.

Data Types

ESSA must support the standard and OS data types, but it can also provide corporative data types. If a data type is normally used by the company’s embedded applications, the ESSA should provide this type along with its methods. This meets the ESSA policy of reusing the code and therefore maturing it.

Hardware Abstraction Level (HAL)

It could be easy to say that it does what its name says, what is the same to say that makes the application level independent of the hardware. Anyway, if a closer look is taken, it can be seen that HAL can be used in other beneficial ways too. As a matter of fact, HAL is not only the natural channel the Tasks or the OS use to interact with the device hardware through the drivers but it can also serve the purpose of improving the possibilities of debugging tasks, drivers and even checking the Processor hardware in the production line.

All these benefits can be achieved if the ESSA is endowed with a System Shell capable of interacting with HAL.

System Shell

The System Shell or ESSA Shell is strongly related to the benefits that Manufacturing and After Sales services processes can get from the Architecture. The System Shell is a group of commands that can be used to get data from the system or to execute actions on it.

Typically, the basic commands are as follows:

Readregister	*WriteRegister*	*ReadRegisterType*
HALdisconnect	*HALConnect*	*Login*
StopApplications	*StartApplications*	*HalpApplications*

Extended and recommended commands could be:

StopSingleTask	*StartSingleTask*	*HaltSingleTask*
HALSingleConnect	*HALSingleDisconnect*	*ShowMeTask*
ShowMeHAL	*ReadMinRegValue*	*ReadMaxRegValue*
GiveMeLastLog	*FlushLog*	*whoami*
Boot	*unboot*	*Hi*
ProductIdentification		

With all the available commands, it is easy to see that the Shell Console can become complex to use, so it is strongly recommended to use a PC compatible Application to perform the system debugging or testing.

Again, as all the properties above are on the same software stack, the code re-usability index and maturity is high.

Finally, System Shell “real world” physical output can vary from a simple RS232 interface to a fast USB port

Task Level

Any Embedded Software independently of its complexity is divided into several modules depending on its functionality. This modular programming is, of course, a well-known good practice that not only eases the

whole implementation process but also gives benefits in terms of software maintenance and code re-usability with a certain set of modifications.

In an ESSA, modules can be understood as independent Tasks and then the modular policy takes even more importance. Here, choosing the right modularity is paramount and there are some questions to be made:

- Can the module become a permanent part of ESSA?
- How important it is to be able to debug the module?
- What is the level of interdependence between the modules?
- What is the module timing?
- Does it need Real Time execution?
- Is there any kind of interaction between the module and the hardware?
- Does the module need a resource that other modules can benefit from?

These and some more questions must be answered before deciding what the application stack should look like in the end and what its relationship with the OS and its resources should be.

In an ESSA environment, Tasks are seen as independent pieces of software that are periodically executed and that have no direct links between them. If communication between Tasks is necessary, the ESSA must provide a way to do so. Anyway, whatever the communication system is, one rule must be taken into account: "If a module is taken away, the software compilation must never crash". Doing so, would mean that there is something in the system that is dependant on that module.

In order to make the software easy-to-port to different hardware platforms, Tasks must avoid any direct reference to the hardware. If interaction with the hardware is needed, it must be done through the HAL, which must provide the methods to do so.

In the end, the aim of all this added complexity is to achieve a system with the following benefits:

- Allow the system to stop/start any tasks for debugging reasons.
- Ease Task Improving/Bug fixing, without affecting the other Tasks.
- Add new features to the system easily by adding just new Tasks that do the job with a minimal system modification.
- Allow Driver/Hardware Testing by stopping the related Task/s and managing the driver/hardware through the system Shell.

Driver Level

The driver level is in charge of allowing the interaction of the system with the outside world. As the way processor peripherals vary from one device to another, drivers must provide a set of standard procedures to allow the Tasks and OS to interact with them through HAL. Standard procedures as initialise, start, stop, read and write are very common but other ones can be added, as for instance readStatus, if information about the status of the peripheral is needed.

What is more, making the OS the one that controls driver start and stop adds the benefit of being able to test the Tasks behaviour by generating fake driver/hardware stimulus through the Shell after stopping the driver.

The Driver Level is the only one that changes from one hardware platform to another, but if a limited number of different devices are normally used, the same number of the driver level stacks can also be developed so they can be reused with the same device repeatedly.

ESSA and PC compatible Applications

From the System Shell section, one can realise the complexity of interacting with the ESSA by command line. For this reason, using a PC compatible Application is highly recommended.

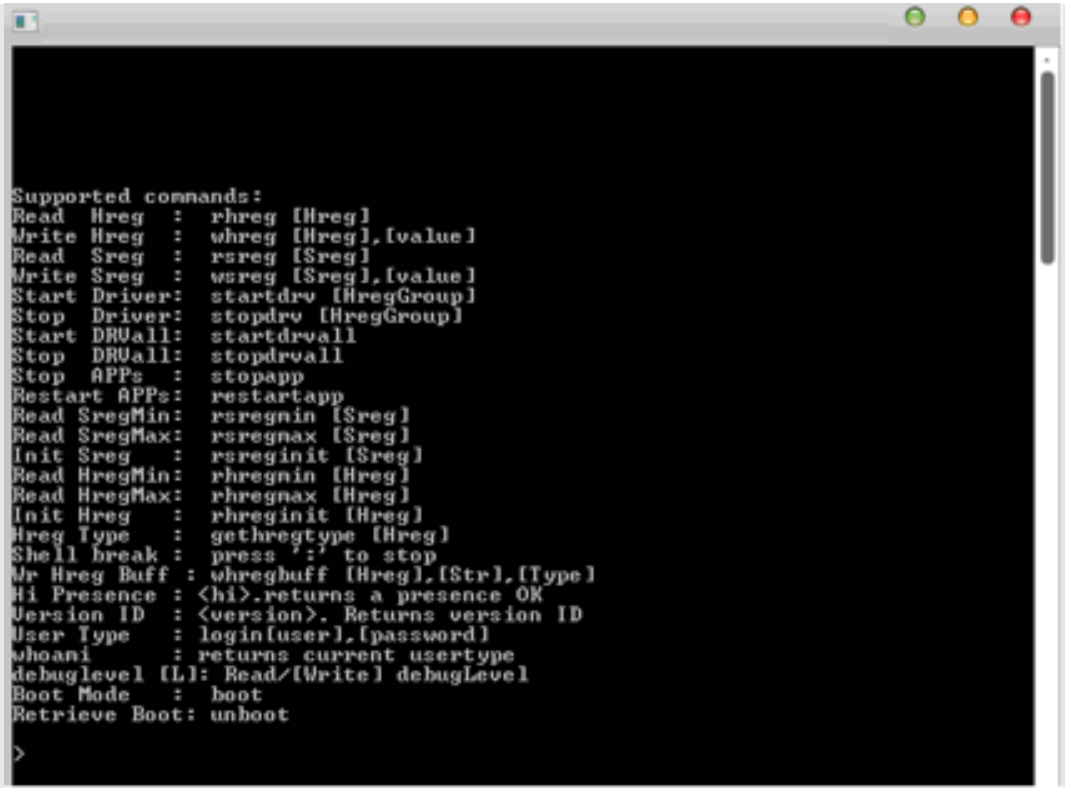

Figure 2: Shell commands example

In that sense, several Already Done applications can be found in the market but to get the most of the Shell possibilities it is again recommended to build an application and customize its performance to satisfy the needs of the Manufacturing and Service processes.

The application will be an easy tool to interact with the ESSA for the following purposes:

- Check the internal data values easily and detect if there is any error.
- Set up the product easily.
- Ease/Automate the backup the of the product set-up.
- Ease/Automate the cloning the of the product set-up.
- Easily sharing of the product set-up (R&D / Manufacturing / After Sales).
- Easily handling and modifying of the product set-up depending on some parameters conditions.
- Example: Modify an internal parameter depending on the other or serial number (Typical After Sales request).
- Store the product set-up easily into the corporate database.

An important issue to mention is that this PC Application is going to be the same within all products based on the same ESSA. The application must be able to detect the product type and then change its own settings to meet the particular product features (e.g. by using a product descriptor file).