AN3423
Application note

Shrinking the AUTOSAR OS: code size and performance optimizations

1 Introduction

This application note describes the operations needed to perform code size and performance optimization of EB's AUTOSAR OS.

EB's AUTOSAR OS is highly configurable. All the standard functionalities of the AUTOSAR OS are available using the standard library right up to Scalability Class 4. The disadvantage of this approach is that the kernel is often much bigger and slower than it needs, instead of using all the techniques to avoid linking unnecessary functions and data into the final binary.

AUTOSAR OS supports the following products:

- SPC564Axx
- SPC560Bxx
- SPC56ELxx
- SPC563Mxx
- SPC560Pxx

Contents

List of tables

2 Optimization

2.1 Library optimization

The AUTOSAR OS is built as a library, it means that functions and data, that are not referenced, are not normally linked into the final binary. The configurability is achieved by making extensive use of decisions based on external (ROM) constants, and function pointers that can be redirected to null (empty) functions. This method of construction means that it is possible to minimize the size of the kernel for customers that do not have a source code license. It means that at each decision there is code for both the true and false cases even though the decision always follows the same branch for a given configuration. For function pointers, the overhead of calling the null function and returning from it is still present.

For customers with a source code license, it is possible to eliminate all the unnecessary decisions and unneeded function calls from the final kernel. The way to achieve it is by building a customized library that is exactly tailored to a given configuration.

The optimizations, with the exception of “API Removal”, are determined from the AUTOSAR OS configuration. Many of optimizations come from the standard configuration, but there is a set of AUTOSAR OS customization options that can result in a smaller, faster kernel with the possible loss of some AUTOSAR conformance.

To be smaller or faster the kernel needs an optimized build. Depending on the target processor and the configuration, an optimized kernel can be as small as around 30% of the size of a standard kernel.

2.2 How to build an optimized library

Using EB's customer build environment it's simply to put OS_BUILD_OPTIMIZED_LIB_FROM_SOURCE = TRUE into the build configuration.It has the following effects:

- the C preprocessor macro OS_USE_OPTIMIZATION_OPTIONS is defined as 1 when compiling all C files. It causes the file os-libconfig.h is included. This file is created by the generator (although a hand-crafted file is not excluded).
- The name of the object-file directory and the kernel library get a library ID encoded into them. This ID identifies exceptionally all the optimizations that affect the kernel library, so that if you change your configuration in a way that changes the optimization a new library is automatically selected and (if necessary) compiled.

If it is using an own build environment, it needs to ensure that C preprocessor macro OS_USE_OPTIMIZATION_OPTIONS is defined to be 1 when compiling all AUTOSAR OS files. Use of a library ID for the library and object files is not compulsory, but if it is used the same name it must delete the libraries and object files and rebuild the library whenever the configuration changes significantly.

“os-libconfig.h” defines several macros, typically called OS_EXCLUDE_something, which tell the kernel that it can omit to code that performs the “something”. The macros are described in the *Section 2.3*.

2.3 Kernel customizations

Kernel customizations are further options that must be explicitly enabled in the AUTOSAR OS configuration. They can provide further reductions in size and runtime. However, many of them result in a kernel whose behavior is not strictly AUTOSAR conformant, so these options must be used with care. In particular, extreme caution must be exercised if customized error handling is selected in a system with protection features enabled.

To use the Kernel Customizations, the OsAutosarCustomization container first must be enabled. Then the options inside the container can be configured as desired.

- OsErrorHandling [MINIMAL, AUTOSAR, FULL]
  - MINIMAL error handling means that API functions simply return an error code if an error is detected. The error hook is not called. Internal errors are be detected but not reported.
  - FULL error handling means that errors in APIs that do not return StatusType are detected and handled. If OsErrorHookForVoidApi is also enabled the ErrorHook appears. The default error action preforms, which could result in the caller being killed.
  - AUTOSAR selects AUTOSAR-conformant error handling. Errors are detected but no action is taken, so the API silently fails to do its job.
- OsErrorHookForVoidApi [TRUE, FALSE]
  - TRUE causes the ErrorHook to be called when an error is detected in an API that does not return StatusType. Needs OsErrorHandling=FULL.
- OsErrorAction [TRUE, FALSE]
  - FALSE disables all special reaction to errors such as terminating tasks and applications. The result is always to return an error code to the caller.
- OsStrictServiceProtection [TRUE, FALSE]
  - Setting this to FALSE disables the very strict calling-checks required by AUTOSAR. The implicit checks that are necessary for correct functioning of the kernel (such as TerminateTask being called from a task) are still present, so this does not affect the safety of the kernel. In the EB kernel, many APIs work correctly when called from a context that is forbidden by AUTOSAR. ActivateTask and SetEvent work correctly when called from an alarm callback or the ErrorHook.
- OsInterruptLockingChecks [MINIMAL, EXTRACHECK,AUTOSAR]
  - MINIMAL means that the interrupt lock status is only checked when it affects the kernel's operation (for example in GetResource and ReleaseResource). EXTRACHECK is for compatibility with ProOSEK - the check is made whenever a task switch is possible. AUTOSAR means full confirmance with the AUTOSAR specification.

Note: In EB's AUTOSAR OS the interrupt lock status is considered to be part of the task's context. Each newly activated task starts with interrupt enabled.

- OsCallIsr [DIRECTLY, VIA_WRAPPER]
  - If ISRs are called DIRECTLY they always run as supervisor with kernel protection boundaries, and they cannot be killed. If a protection error occurs in an ISR the only possible course of action is SHUTDOWN.
- OsCallAppErrorHook [DIRECTLY, VIA_WRAPPER]
  - If application-specific ErrorHooks are called DIRECTLY they always run as supervisor with kernel protection boundaries, and they cannot be killed. If a protection error occurs in a hook function the only possible course of action is SHUTDOWN.
- OsCallAppStartupShutdownHook [DIRECTLY,VIA_WRAPPER]
  - If application-specific startup and shutdown hooks are called DIRECTLY they always run as supervisor with kernel protection boundaries, and they cannot be killed. If a protection error occurs in a hook function the only possible course of action is SHUTDOWN.
- OsPermitSystemObjects [TRUE,FALSE]
  - In a system with OS-Applications, enabling this feature permits OS-objects to belong to “the system” and not to any particular application. Such objects can “access” other objects without restriction. This feature is useful for objects such as schedule tables that control the scheduling of all applications in a system.
- OsUserTaskReturn [KILL_TASK,LOOP]
  - This option determines what happens when a task returns from its main function. KILL_TASK is AUTOSAR-conformant but requires the full error handling and error action to be enabled for correct functionality. LOOP means that if a task returns from its main function it tries to terminate. If that fails it shuts down the AUTOSAR OS. If that fails it enters an endless loop, which has the effect of locking out all tasks of equal or lower priority.

Note: On architectures such as Tricore where return from-task is caught using a special exception handler, this option has no effect.

2.4 API removal

API removal is most important for kernels that use an hardware trap mechanism (called a “system-call”) for calling kernel functions. On the Power Architecture, use of the system-call is a configurable option.

When a system-call is not used, the user's code calls kernel functions directly. This means that kernel functions that are not used do not get selected from the library at link time. With a system-call, each call to a system service results in a system-call with a unique index, and the index is used by the system-call handler to select the appropriate kernel function from a table.

In a kernel with system-calls, many kernel functions can be eliminated automatically. For example, if there are no Resources defined, the GetResource and ReleaseResource functionality can be excluded. However, it is not possible to determine from the configuration whether some kernel functions are used or not. A good example is ChainTask(), which can be quite large and is seldom used.

To remove the macro not used, the OS_USE_API_REMOVAL should be defined. This can be achieved by defining it on the compiler command line using -DOS_USE_API_REMOVAL. Defining this macro causes the kernel to include the file Os_api_removal.h (was: os-remove-api.h). This file must be provided by the user, and contains a set of macro definitions of the form OS_EXCLUDE_<api-name>, where <api-name> is the name of the API function to exclude, all capitalised. For example, to exclude the ChainTask API you would define the macro OS_EXCLUDE_CHAINTASK:

```
#define OS_EXCLUDE_CHAINTASK 1
```

2.4.1 What happens if a removed API is called?

In kernel without system calls, the API would work normally. In a kernel with system calls, the API call is routed to the error handler with the internal code OS_ERROR_UnknownSystemCall. This results in an E_OS_VALUE error. The error hook is called if configured, and the API returns E_OS_VALUE. In STANDARD status the caller is likely to be terminated.

2.5 Interrupt locking API

The six interrupt locking APIs are by default implemented using direct function calls. This means that they can only be used by trusted applications, because non-trusted applications typically run in a restricted processor mode where access to the registers and instructions for manipulating interrupt locks is restricted. The system-call versions of these APIs are still present, however, and can used by non-trusted code.

If it is known that the interrupt locking system calls are not being used, the six system calls can be omitted by defining these macros:

```
#define OS_EXCLUDE_DISABLEALLINTERRUPTS 1
#define OS_EXCLUDE_ENABLEALLINTERRUPTS 1
#define OS_EXCLUDE_SUSPENDALLINTERRUPTS 1
#define OS_EXCLUDE_RESUMEALLINTERRUPTS 1
#define OS_EXCLUDE_SUSPENDOSINTERRUPTS 1
#define OS_EXCLUDE_RESUMEOSINTERRUPTS 1
```

2.6 Optimization options

All the optimization recognized by the kernel are obtained directly from the AUTOSAR OS configuration by the generator and should under no circumstances be defined by hand. These optimizations are recorded in the following list:

- OS_EXCLUDE_APPLICATIONS

It is defined if the configuration contains no OS-Applications. Many application related functions can be omitted.

- OS_EXCLUDE_CALLINGCONTEXTCHECK

Removes the explicit calling-context check from kernel API functions.

- OS_EXCLUDE_CAT2ISR

The configuration contains no category 2 ISRs. Some functions related to ISRs can be omitted.

- OS_EXCLUDE_CPULOAD

The CPU load monitoring feature is omitted.

- OS_EXCLUDE_ERROR_ACTION

No special error action is taken. Errors always result in an error code being returned to the caller.

- OS_EXCLUDE_ERRORHANDLING

The complete error handling code is omitted. Error codes are returned to the caller. No hook functions are called.

- OS_EXCLUDE_ERRORHANDLINGFORVOIDAPI

Error handling for APIs does not return, an error code is omitted (it is the default).

- OS_EXCLUDE_ERRORHOOK

The code that calls the error hook is omitted.

- OS_EXCLUDE_ERRORHOOK_APP

The code that calls the application's error hook (including processor mode switch) is omitted.

- OS_EXCLUDE_EVENTS

All functions related to Events (WaitEvent, SetEvent, etc.) are omitted. A few other optimizations in ActivateTask are also possible.

- OS_EXCLUDE_EXCEPTIONS

The standard exception handling is omitted. In its place there is a call to an userprovided function.

- OS_EXCLUDE_EXTENDED

All error checking that takes place in EXTENDED status is omitted.

- OS_EXCLUDE_EXTRA_CHECK

All code related to “extra runtime checks” is omitted.

- OS_EXCLUDE_HWCOUNTER

All the functionality for hardware counters (initialisation, starting and stopping during alarm processing) is omitted.

- OS_EXCLUDE_HW_FP

Architecture-dependent: floating-point context switching is omitted.

- OS_EXCLUDE_INTSENABLEDCHECK

The check that interrupts are enabled, that AUTOSAR requires in almost all API functions, is omitted.

- OS_EXCLUDE_KILLABLE_APPEHOOK

Error hooks belonging to applications are called directly without saving context. This means that they run as trusted code and cannot be killed.

- OS_EXCLUDE_KILLABLE_APPSHOOK

The Startup and Shutdown Hooks belonging to applications are called directly without saving context. This means that they run as trusted code and cannot be killed.

- OS_EXCLUDE_KILLABLE_ISR

Interrupt service routines are called directly without saving context. This means that they run as trusted code and cannot be killed.

- OS_EXCLUDE_KILLALARM

The function for killing an alarm is omitted. It means an application with an active alarm cannot be properly terminated.

- OS_EXCLUDE_KILLISR

The function for killing an ISR is omitted. It means an ISR can never be killed in response to a protection error.

- OS_EXCLUDE_KILLTASK

The function for killing a task is omitted. It means a Task cannot be killed in response to a protection error.

- OS_EXCLUDE_MULTIPLE_ACTIVATIONS

If there are no tasks with multiple activations, quite a lot of code in ActivateTask and TerminateTask can be omitted.

- OS_EXCLUDE_PARAMETERACCESS

If the error hooks do not need to determine what API parameters cause the error, the code that passes the parameters through the error handling can be omitted. It affects all API functions.

- OS_EXCLUDE_POSTISRHOOK

The code that calls the PostISRHook is omitted.

- OS_EXCLUDE_POSTTASKHOOK

The code that calls the PostTaskHook is omitted.

- OS_EXCLUDE_PREISRHOOK

The code that calls the PreISRHook is omitted.

- OS_EXCLUDE_PREEMPTION

Architecture-dependent: the possible context switch at the end of the interrupt handler can be simplified.

- OS_EXCLUDE_PRETASKHOOK

The code in the error handler that calls the PreTaskHook is omitted.

- OS_EXCLUDE_PROTECTION

All code related to memory protection is omitted.

- OS_EXCLUDE_PROTECTIONHOOK

The code in the error handler that calls the ProtectionHook is omitted.

- OS_EXCLUDE_RATEMONITORS

All the arrival rate limiting code in ActivateTask, SetEvent, WaitEvent and in the handling of Category 2 ISRs is omitted.

- OS_EXCLUDE_RESOURCEONISR

The code that adjusts the interrupt levels in GetResource and ReleaseResource is omitted.

- OS_EXCLUDE_SHUTDOWNHOOK

The code that calls the ShutdownHook is omitted.

- OS_EXCLUDE_STACKCHECK

All software stack checking is omitted.

- OS_EXCLUDE_STARTUPHOOK

The code that calls the StartupHook in StartOS is omitted.

- OS_EXCLUDE_SWCOUNTER

All code related to Software Counters (including the IncrementCounter API) is omitted.

- OS_EXCLUDE_TIMINGPROTECTION

All the code that implements execution-time protection (execution budget, resource and interrupt lock timing) is omitted.

- OS_EXCLUDE_USERTASKRETURN

The code that handles a return from a task's main function is omitted. If a task fails to call TerminateTask and simply returns from its main function, the result is undefined but most likely result in the task entering an endless loop, which lock out equal and lower priority tasks.

- OS_EXCLUDE_AGGREGATELIMIT

No longer used. The aggregate execution-time limit was removed in AUTOSAR 3.0.

2.7 Disadvantages

The main disadvantage of using the custom kernel is the build time. Building a kernel library under Windows with EB's customer “make” environment can take a good few minutes - although building on Linux with a simplified make environment can be achieved in under 10 seconds. If compile time is a problem, it is better to use a standard library in the early stages of a project, when the configuration is undergoing change. If optimization is used extensively while the configuration is changing, the AUTOSAR OS' library directory fills up with the different library versions and the corresponding object files, but these can be deleted if disk space becomes a problem.

Appendix A Acronyms

Table 1. Acronyms

Acronym	Name
EB	Elektronic Bit
ECU	Electonic Control Unit
API	Application Programming Interface

Revision history

Table 2. Document revision history

Date	Revision	Changes
14-Jul-2011	1	Initial release.
18-Sep-2013	2	Updated Disclaimer.