HYUNDAI AUTOEVER

AUTOSAR Os Profiler User Manual

DOC. NO

SCOPE OF APPLICATION All Project/Engineering
Responsibility : Classic AUTOSAR Team

File Name Os_Profiler_UM.docx
Creation YongHyun Han
Check HoiMin Kim
Approval InWon Kang
Edition Date: 2023/03/17
Document Management System

HYUNDAI AutoEver

Document Change Histroy				
Date (YYYY-MM-DD)	Ver.	Editor	Chap	Description (before -> after revision)
2016-07-19	1.0.0	SH.Yoo		• Initial Creation
2016-09-02	1.0.1	SH.Yoo	3.2 4.2 5.1.1 All	• Modified a term 'Isr' to 'Category2 Isr' • Modified version information • Fixed wrong reference information •Fixed wrong or strange expressions
2016-09-09	1.0.2	SH.Yoo	4.3.2 5.1.1 9.1	• Added a limitation item • Deleted PM configuration •
2016-10-28	1.0.3	SH.Yoo	4.2 4.3.2	• Modified a limitation item •
2016-12-01	1.1.0	SH.Yoo	4.3.2 5.1.2 6.3.6	• Modified a limitation item • Modified a prototype of Opf_GetTaskInfo •
2017-03-28	1.1.1	JH.Lee	4.3.1	• Fixed warning
2017-08-29	1.1.2	MJ.Woo	4.3.2 5.1.1 5.1.2	• Added TC2xx support •
2018-03-09	1.2.0	YH.Han	4.3.2 5.1.2 6.1.3 6.1.4 8.6 8.7 8.8 8.9	• Added SPC58xx support version at limitation • Added supporting SPC58xx and TC2xx at TargetOperatingSystem • Added description of Tdd_Opf_ProfileTaskInfo and Tdd_Opf_ProfileEventQ • Added description of max Jitter, max CET, max GET and max RT
2018-11-19	1.3.0	YH.Han	4.3.1 4.3.2 5.1.1. 8.3 8.10	• Added TC27x Multi-Core Support •
2019-05-23	1.4.0	YH.Han	4.3.1 4.3.2 5.1.2	• Added S32K1xx Support
2019-11-13	1.5.0.0	YH.Han	4.3.1 4.3.2 5.1 8.10	• Added new configurator for AUTRON_COMMON
2020-06-17	1.5.1.0	YH.Han	4.3.1 4.3.2 5.1	• Changed category of some configuration items • Changed StmFrequency to input KHz unit • Changed limitation for s32k (change default timer LPIT -> FTM)
2020-11-30	1.5.2.0	YH.Han	4.3.1	• Released OsProfiler-1.5.2.0

2021-01-13	1.5.3.0	MJ.Woo	4.3.1	• Released OsProfiler-1.5.3.0
2021-12-21	1.5.4.0	YH.Han	4.3.1	• Released OsProfiler-1.5.4.0
2022-02-18	1.5.5.0	JC.Kim	4.3.1	• Released OsProfiler-1.5.5.0
2022-03-23	1.5.6.0	YH.Han	4.3.1	• Released OsProfiler-1.5.6.0
2022-04-05	1.5.7.0	YH.Han	4.3.1	• Released OsProfiler-1.5.7.0
2022-07-04	1.5.7.1	HJ.Kim	1 4.3.1	• Clarified copyright of the code • Released OsProfiler-1.5.7.1
2022-08-10	1.5.8.0	JC.Kim	4.3.1	• Released OsProfiler-1.5.8.0
2023-03-17	1.5.8.1	YH.Han	4.3.1	• Released OsProfiler-1.5.8.1

Table of contents

1. Overview

This document is created based on the OS Profiler SRS. If more detailed functional description is required in using the OS Profiler module, please refer to the reference documents.

Each configuration category is defined as follows.

- Changeable (C): Items that can be configured by users
- Fixed (F): Items that cannot be changed by users
- NotSupported (N): Items that are not used

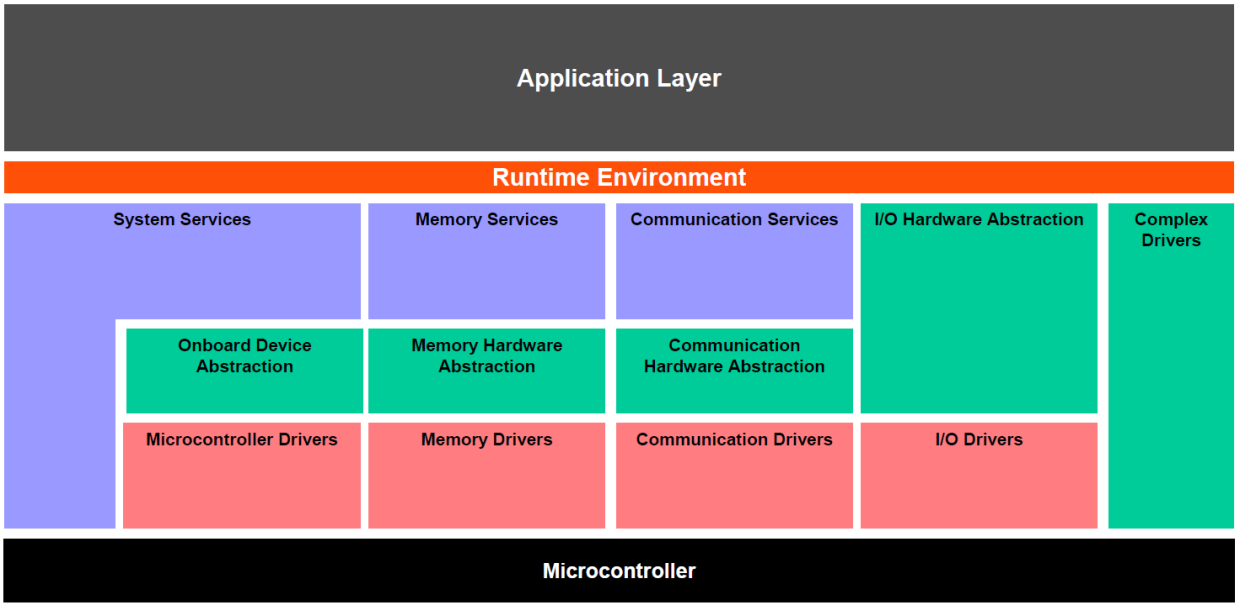
2. Reference

Sl. No.	Title	Version
1.	Os_Profiler_ESRS.docx	1.5.0

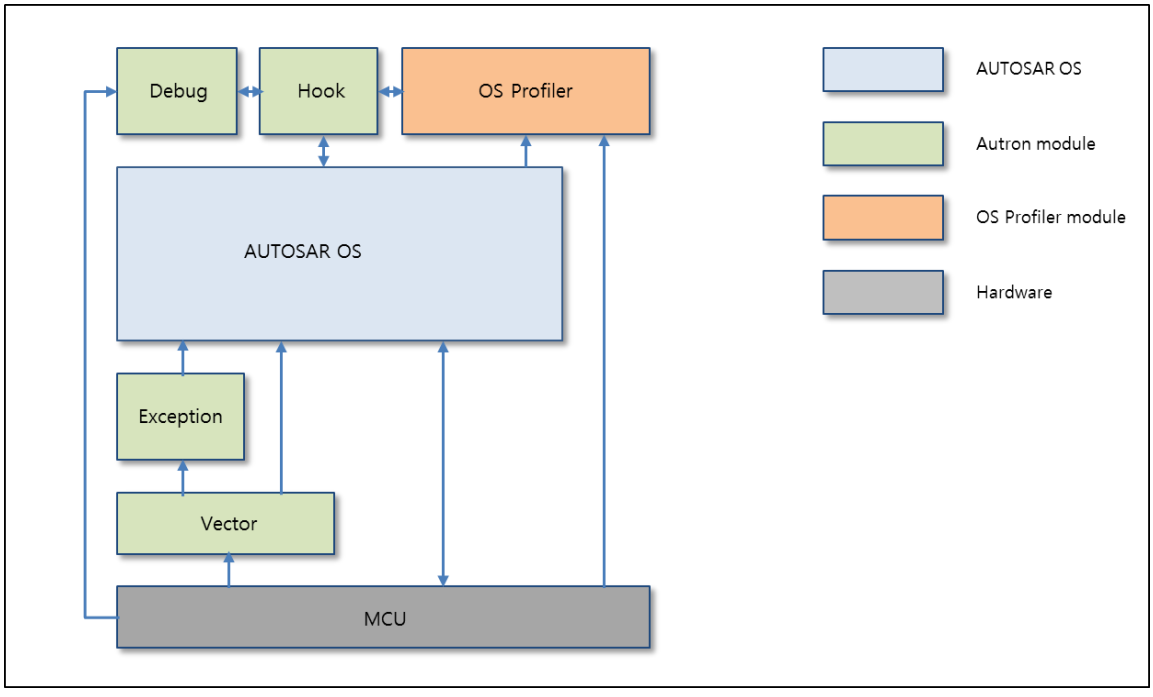
3. AUTOSAR System

3.1 Overview of Software Layers

The AUTOSAR platform is a layered architecture as illustrated below. The AUTOSAR platform can be divided into Service Layer, ECU Abstraction Layer, Complex Device Drivers and Microcontroller Abstraction Layer.



AUTOSAR OS belongs to System Services, and its structure is as the diagram below. On the basis of the AUTOSAR OS, AutoEver modules such as Vector, Exception, Debug and Hook are added to complete the whole AUTOSAR OS module. In this illustration, the OS Profiler collects tasks and Category2 ISR events based on AUTOSAR OS.



3.2 OS Profiler

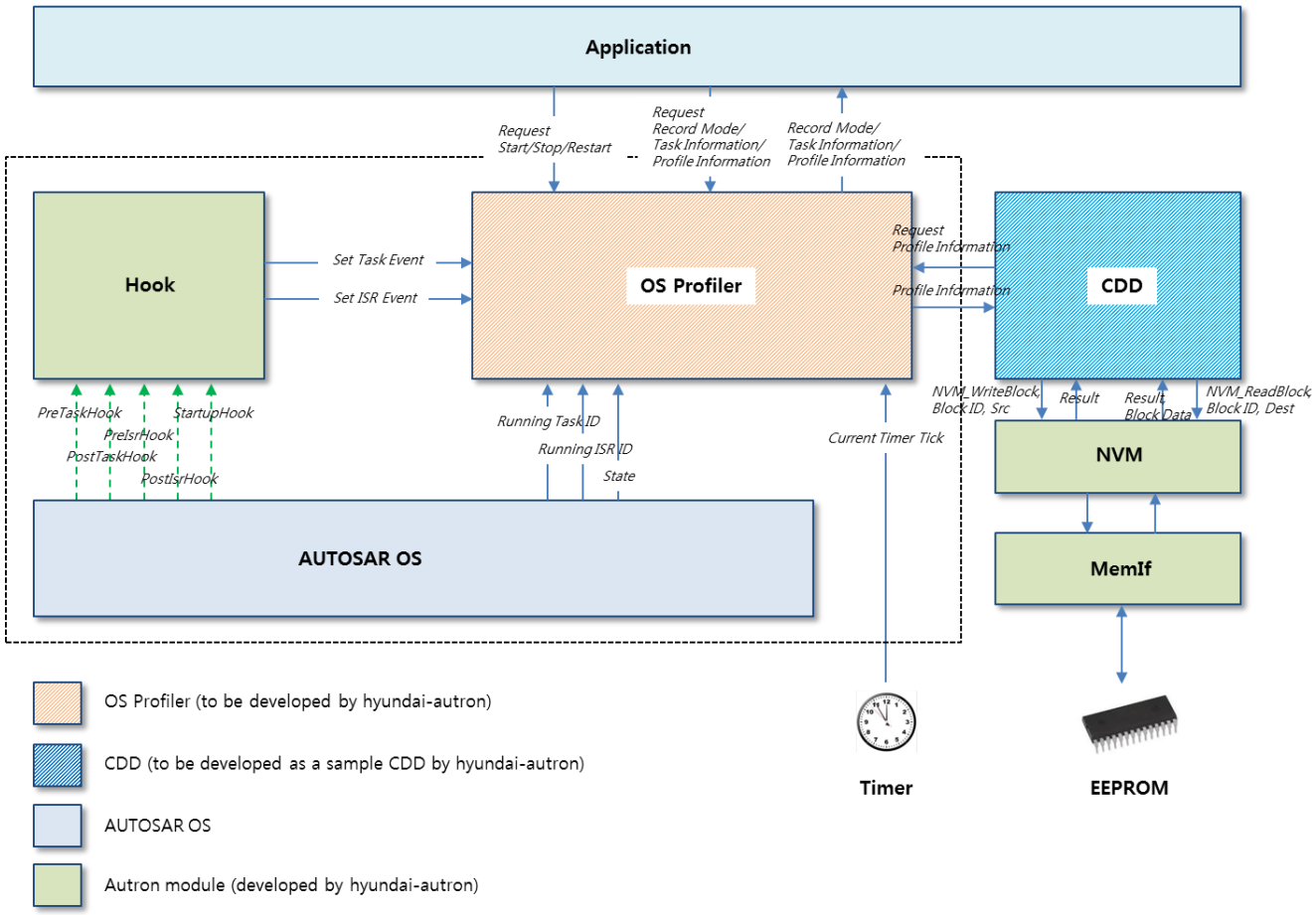
The OS Profiler module collects periodic tasks and Category2 ISR state information real-time, and the user can gather the information on the behaviors of the tasks and Category2 ISR based on that. Especially, it can be estimated whether a periodic task can enter Running* state on exact timing based on information about other tasks before the periodic task and Category2 ISR.

To summarize, the key functionalities of the OS Profiler module are as follows:

- Collecting state information of tasks and Category2 ISR
- Collecting "idle state**" information of the MCU core prior to a periodic task

* Running: a task state in which the task is actually running, occupying MCU resource.
* Idle state: a state in which the MCU core is not doing anything. From Idle, a task or ISR can directly be run (in case of a task, it transitions to the Running state).

The OS Profiler module is connected to software modules – Application, Hook, AUTOSAR OS, and CDD – and hardware module Timer.



4. Product Release Notes

4.1 Overview

This chapter provides the release information of Hyundai AutoEver OS Profiler, describing the features and restrictions of different versions of OS Profiler software product.

4.2 Scope of the release

All content in this document applies only to the following Hyundai AutoEver OS Profiler module.

Module	Module version
OS Profiler	1.5.8

※ Module version refers to the SW version of the BswModule Description (Bswmd) file of each module.

4.3 Module release notes

4.3.1 Change Log

4.3.1.1 Version 1.5.8.1

➢ **Improvements**

- An english Um document added

Cause	Request for English UM document
Operation effect	None
Setting effect	None
ASW Action	None

4.3.1.2 Version 1.5.8.0

➢ **Improvements**

- Code improvement to comply with the UNECE Cyber Security regulations.

Cause	Customer request
Operation effect	None
Setting effect	None
ASW Action	None

4.3.1.3 Version 1.5.7.1

➢ **Improvements**

- Clarify the copyright of code in E-code, Generate Code

Cause	Change request about copyright
Operation effect	N/A
Setting effect	N/A
ASW Action	N/A

- Content file DeliveryBoxHistory modification

Cause	Change request to apply new template
Operation effect	N/A
Setting effect	N/A
ASW action	N/A

4.3.1.4 Version 1.5.7.0

➢ **Improvements**

- Resolved the link error that occurs when EnableOsProfiler setting is false.

Cause	The link error occurred when the aforementioned setting was false because of the folder structure changes in OsProfiler-1.5.4.0.
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.5 Version 1.5.6.0

➢ **Improvements**

- Code improvement to comply with the UNECE Cyber Security regulations.

Cause	Violation of UNECE Cyber Security regulations occurred
Operation effect	None
Setting effect	None
ASW action	None

- Modified the generator so that the input file list is sorted in the comments inside the arxml file of the output.

Cause	Corrected the issue in which the file order in the arxml file came out different every time files were generated.
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.6 Version 1.5.5.0

➢ **Improvements**

- Corrected an issue in which some variables pointed to incorrect memory sections during linking.

Cause	Fixed incorrect designation of memory sections for some variables
Operation effect	None
Setting effect	None
ASW action	None

- Corrected an issue in which E_OS_ID ErrorHook occurred due to RH850 Object ID type mismatching.

Cause	Corrected incorrect type declarations of variables related to ObjectID
Operation effect	E_OS_ID error will not occur.
Setting effect	None
ASW action	None

- Fixed an issue in which a build error occurred due to incorrect syntax in Bolero API extern declarations.

Cause	The type of input parameter was omitted in the Bolero MPC560XB MCU API extern declaration.
Operation effect	Build Error (error #92) will not occur.
Setting effect	None
ASW action	None

4.3.1.7 Version 1.5.4.0

➢ **Improvements**

- Code improvement to comply with the UNECE Cyber Security regulations.

Cause	Violation of UNECE Cyber Security regulations occurred
Operation effect	None
Setting effect	None
ASW action	None

- Applied a new document template

Cause	The template has been changed due to company merger.
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.8 Version 1.5.3.0

➢ **Improvements**

- Addressed or justified static verification violations

Cause	It was necessary to reflect the results of static verification.
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.9 Version 1.5.2.0

➢ **Improvements**

- Enabled octal input in the generator.

Cause	Octal values were incorrectly recognized for parameters that allows octal, decimal, and hexadecimal input.
Operation effect	None
Setting effect	None
ASW action	None

- Addressed or justified static verification violations

Cause	Static verification was necessary
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.10 Version 1.5.1.0

➢ **Improvements**

- Allowed KHz input for OsProfiler StmFrequency configuration values (specify KHz values in places below decimal point).

Cause	MHz input doesn't work for some MCUs whose timer accepts frequency vales in KHz.
Operation effect	None
Setting effect	None
ASW action	None

- Modified some configuration categories

Cause	Modified categories of MCU-dependent items
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.11 Version 1.5.0.0

➢ **Features**

- Added new AUTRON_COMMON OsProfiler configurations

Cause	To eliminate MCU dependency of OsProfiler
Operation effect	None
Setting effect	Depending on MCU, OsProfilerGlobalConfig, OsProfilerCommonTargetConfig, and OsProfilerSpecificTargetConfig need to be configured.
ASW action	None

- Aurix TC3xx 6 core support

Cause	Demand for OS Profiler support on Aurix TC3xx 6 Core
Operation effect	None
Setting effect	Depending on the number of cores on which OsProfiler is run, it is necessary to configure OsProfilerGlobalConfig > NumberOfCores.
ASW action	None

➢ **Improvements**

- Added count break for OsProfiler's TimeIndex iteration.

Cause	Exception handling for cases where the system spent too much time in the iterations unnecessarily during TimeIndex processing.
Operation effect	None
Setting effect	None
ASW action	None

- Modified configuration categories

Cause	Changed some items in the Fixed category to Changeable
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.12 Version 1.4.0

➢ **Features**

- Supports S32K1xx MCU

Cause	Demand for OS Profiler support on S32K1XX MCU
Operation effect	None
Setting effect	None
ASW action	None

➢ **Improvements**

- Corrected compile warnings

Cause	Compile warnings occurred due to unreachable code on MCUs that do not support multicore.
Operation effect	None
Setting effect	None
ASW action	None

➢ **Improvements**

- Addressed static verification violations

Cause	Corrected static verification violations including unreachable code due to user configurations.
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.13 Version 1.3.0

➢ **Features**

- Aurix Multi-Core support

Cause	Demand for OS Profiler support on Aurix Multi-Core
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.14 Version 1.2.0

➢ **Features**

- Supports ST SPC58xx MCU

Cause	Demand for OS Profiler support on SPC58xx MCU
Operation effect	None
Setting effect	None
ASW action	None

➢ **Improvements**

- Correct an issue that incorrectly displayed the period value of tasks.

Cause	Opf_SaveTimeStamp() was omitted when OPF_PROFILE_ISR == STD_OFF during the modification of Opf_SetPreIsrEvent in OsProfiler 1.1.2.
Operation effect	None
Setting effect	None
ASW action	None

- Added content for user manual

Cause	Added lacking content and modified errors on previous versions. - 5.1.2: added TC2xx in TargetOperatingSystem - 6.1.3, 6.1.4: Added descriptions for Tdd_Opf_ProfileTaskInfo and Tdd_Opf_ProfileEventQ - 8.6 - 8.9: added descriptions for max jitter, max CET, max GET, and max RT
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.15 Version 1.1.2

➢ **Features**

- To support Infineon TC2xx MCU

Cause	Demand for OS Profiler support on TC2xx MCU
Operation effect	None
Setting effect	None
ASW action	None

➢ **Improvements**

- Corrected an issue where the information of TaskInfo is mistakenly shown for a different task.

Cause	When more than 10 instances of OsProfilerPeriodTaskConfig are configured, the connections between Opf_GaaProfileTaskInfo[x] and Opf_GaaTaskInfo[x] were generated misaligned by the Generator.
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.16 Version 1.1.1

➢ Improvements

Improved warnings.

Cause	Warnings occurred because unnecessary calculations were conducted.
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.17 Version 1.1.0

➢ Features

- N/A

➢ Improvements

- Corrections made to use integer instead of floating in some calculations

Cause	Corrections made to use integer instead of floating when calculating lowest waiting margin and jitter (so that an additional library for floating calculation is unnecessary)
Operation effect	Minimum waiting margin and jitter are now per mille values instead of per cent.
Setting effect	None
ASW action	ASWs using lowest waiting margin and jitter should be updated since they are now per mille integer values.

- Addressed a library deployment issue.

Cause	An issue resulting from the difference in configurations between when the library was deployed and when it was utilized.
Operation effect	None
Setting effect	None
ASW action	None

- Improved the timing of checking Task Activation.

Cause	It was necessary to change the timing of checking Task Activation from after counter processing to before.
Operation effect	None
Setting effect	None
ASW action	None

- Improved function Opf_GetTaskInfo.

Cause	Fixed an issue in which Task ID from OS could not be used directly as argument.
Operation effect	None
Setting effect	None
ASW action	Task ID provided by the OS can now be directly used as an argument of Opf_GetTaskInfo without any conversion.

- Corrected an issue in which E_OS_LIMIT events were not recorded on RH850.

Cause	There was an issue in which E_OS_LIMIT events were not recorded on RH850.
Operation effect	None
Setting effect	None
ASW action	None

- Fixed an issue in which state information of delayed tasks was not refreshed on RH850

Cause	When an immediate context switch occurred after PostTaskHook to another task and an interrupt was triggered during the transition, PreIsrHook was called instead of PreTaskHook.
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.18 Version 1.0.3

➢ **Features**

- N/A

➢ **Improvements**

- Improved measurement of the range of PLL m-factor value on RH850.

Cause	Timer Clock was not correctly measured due to miscalculations of the range of PLL m-factor value on RH850.
Operation effect	Timer Clock is not correctly measured if a certain m-factor value is used.
Setting effect	None
ASW action	None

- Updated User Manual.

Cause	Statement that ISR events are not supported on RH850 OS (Chapter 4.3.2)
Operation effect	None
Setting effect	None
ASW action	None

4.3.1.19 Version 1.0.2

➢ **Features**

- N/A

➢ **Improvements**

- Updated User Manual.

Cause	It was necessary to add statements on mass production and remove content on PM configurations. (Chapter 4.3.2, 5.1.1, 9.1)
Operation effect	None
Setting effect	None
ASW action	None

4.3.2 Limitations

1. OS Profiler runs on AUTOSAR OS and is therefore dependent on it. OS Profiler currently supports AUTOSAR OSes listed below.

- **Freescale AUTOSAR OS/MPC5600 v.4.0**
- **RX-AUTOSAR850 V2, RV850 V2.01.03**
- **AUTOEVER AUTOSAR OS TC2xx v2.4.0 or later**
- **AUTOEVER AUTOSAR OS TC3xx v0.3.0 or later**
- **AUTOEVER AUTOSAR OS SPC58xx v2.2.0 or later**
- **AUTOEVER AUTOSAR OS S32K1xx v1.2.0 or later**
- **AUTOEVER AUTOSAR OS S32K1xx v1.2.0 or later**
- **AUTOEVER AUTOSAR OS CYTxxx v1.1.0 or later**

2. ISR event information (entering and terminating ISR) collected by OS Profiler is limited to Category2 ISR. Category1 ISR events are not collected by OS Profiler.

3. The type of tasks subject to profiling by OS Profiler is limited to periodic tasks. It is because it is required to identify when the task was activated to collect information necessary for task profiling and it is impossible to find out the timing of activation for tasks that are activated by means other than an alarm (e.g. ActivateTask API).

4. The status of OS should be set to EXTENDED.

5. Multicore environments are partially supported. Currently, multicore OS Profiler is supported on Aurix. It is supported for up to 3 cores on TC2xx and for up to 6 cores on TC3xx. OS Profiler properly functions related to the behavior of each core in an environment being synchronized at the same STM frequency.

6. On Freescale OS, only HWCOUNTER and SWCOUNTER are supported based on the System Timer (Second Timer or user-defined SW counter is not supported).

7. This functionality must not be included in the mass production version of the software (EnableOsProfiler in OS Profiler Configuration must be set to false).

8. S32K1xx systems support OS Profiler functionalities based on FTM. The 16-bit counter of FTM has a limitation on the amount of time it can express. Therefore, the Time Index Period of OsProfiler should be set to a value lower than [2^16 / StmFrequency]. The period of periodic tasks that can be measured by OsProfiler is also limited to [2^16 / StmFrequency].

9. OsProfiler on S32K1xx uses FTM3 by default; the unit is used alongside an internal or external watchdog. Therefore, when configurations related to FTM3 are modified in OsProfiler or an internal or external watchdog, other modules may be affected.

4.3.3 Deviation

None

5. Configuration Guide

5.1 OS Profiler Module

5.1.1 OsProfilerGlobalConfig Container

In this container, generic functionalities required for OS Profiler to function are configured.

Parameter Name	Value	Category
EnableOsProfiler	true/false	C
EnableProfileIsr	true/false	C
NumberOfCores	User Defined	C
ProfileEventCount	User Defined	C
EnableTimeIndex	true/false	C
UpwardDirectionCount	true/false	F
TimeIndexPeriod	User Defined	C
TargetOperatingSystem	User Defined	F

1. EnableOsProfiler

- Turn on or off the functionalities of OS Profiler.
  - true: enable OS Profiler functionalities
  - false: disable OS Profiler functionalities

- This item must be set to false in the final, mass-production version software since it doesn't include OS Profiler.

2. EnableProfilerIsr

- Determines whether to collect Category2 ISR events.
  - true: collect Category2 ISR events
  - false: doesn't collect Category2 ISR events

3. NumberOfCores

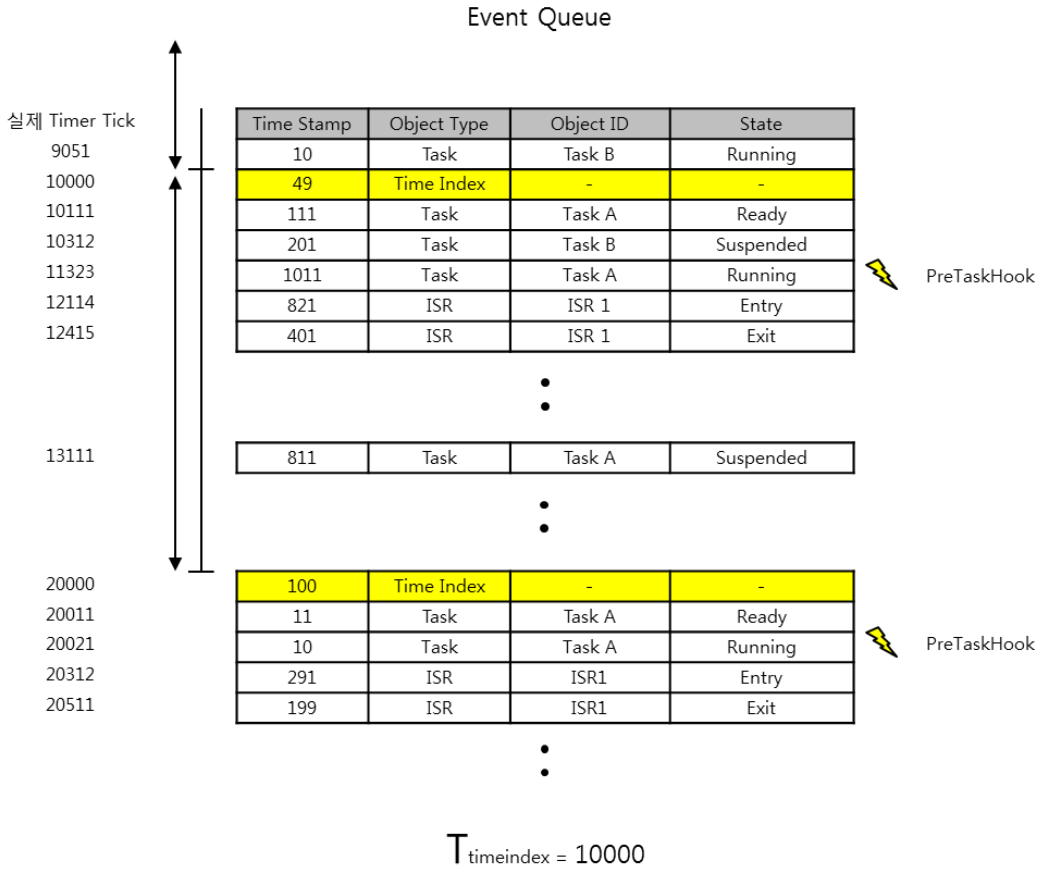
- Set the number of cores on which OS Profiler will function. The value can be 1 to 3 for TC2xx, 1 to 6 for TC3xx. For other MCUs, "1" is the only valid configuration. This applies to cores with lowest numbers first. i.e. when it is set to 5, Os Profiler will run on Core0 to Core4.

4. ProfileEventCount

- Sets the number of Task/Cat2ISR events to be profiled. The maximum value of profile events allowed for OS Profiler is 32,767. An event occupies 24 bytes of memory space (12 bytes for event queue and 12 bytes for profile event queue); the maximum number of profile events should be set within the limits of the system memory.

5. EnableTimeIndex

- Determines whether to collect TimeIndex events. As follows, TimeIndex is stored in the event queue in accordance with the period configured by the user.



$T_{timeindex}$ = 10000

- true: collect TimeIndex events
- false: doesn't collect TimeIndex events

6. UpwardDirectionCount

- Set the order in which the counter (referenced by OsProfiler) counts.
  - true: ascending, i.e. the counter increases starting from zero.
  - false : descending, i.e. the counter decreases starting from the max value.

7. TimeIndexPeriod

- Sets the period of TimeIndex events.

8. TargetOperatingSystem

- Specifies the OS used by the current system. Set to FREESCALE_MPC560xB when using Freescale's MPC560xx; to RENESAS_RH850 when using Renesas' RH850. Select AUTRON_COMMON for any other.

5.1.2 OsProfilerSpecificTargetConfig Container

This container is referenced when TargetOperatingSystem is FREESCALE_MPC560xB or RENESAS_RH850.

Parameter Name	Value	Category
XOSC	User Defined	C

1. **XOSC**

- Specifies the external oscillator clock used by the ECU. This value is used to calculate the System Timer Clock that is used by OS Profiler.

5.1.3 OsProfilerCommonTargetConfig Container

This container is referenced when TargetOperatingSystem is AUTRON_COMMON.

Parameter Name	Value	Category
StmAddress	User Defined	F
StmMaxTick	User Defined	F
StmFrequency	User Defined	F

1. **StmAddress**

- Sets the address of the timer referenced by OS Profiler.

2. **StmMaxTick**

- Sets the maximum expressible value of the timer referenced by OS Profiler.

3. **StmFrequency**

- Sets the frequency of the timer referenced by OS Profiler in MHz. Use a decimal point to specify a value by KHz.

5.1.4 OsProfilerPeriodTaskConfig Container

This container configures periodic tasks to be profiled by OS Profiler.

Parameter Name	Value	Category
PeriodTaskRef	User Defined	C
TaskPeriod	User Defined	N

1. **PeriodTaskRef**

- Sets a reference to the task that will be profiled. The task to be referenced will be the periodic tasks that are already configured in the OS.

2. **TaskPeriod**

- Sets the period of the task that will be profiled in seconds. This value is currently unused.

6. Application Programming Interface (API)

6.1 Type Definitions

6.1.1 OpfRecModeType

Type:	Scalar
Range	0 ~ 3
Description:	Data type for a Record Mode of OS Profiler Enumeration ● OPF_RECMODE_STOP (0) ● OPF_RECMODE_EVENT (1) ● OPF_RECMODE_PROFILE (2) ● OPF_RECMODE_RESTART (3)

6.1.2 OpfTaskIDType

Type:	Scalar
Range	0 ~ 255
Description:	ID of Task

6.1.3 Tdd_Opf_ProfileTaskInfo

Type:	Structure
Range	-
Description:	This structure holds an information of each profiled task. OsProfiler stores worst execution time such as min ST, max jitter, max CET, max GET and max RT of each tasks. Please refer to chapter 8 for more details. ddTaskID : Task ID ddTaskPeriod : Period of Task ddMinST : Min Slack Time ddMaxJIT : Max Jitter ddMaxCET : Max Core Execution Time ddMaxGET : Max Gross Execution Time ddMaxRT : Max Response Time pTaskInfo : Pointer to a task information

6.1.4 Tdd_Opf_ProfileEventQ

Type:	Structure
Range	-
Description:	This structure holds an information of each profiled event queue. OsProfiler stores event queue information of the shortest Slack Time. Please refer to chapter 8.4 for more details. ddEventQueueSize : Size of an event queue ddLastEventPos : Position for a last event in an event queue pCurProfiledTaskInfo : Pointer to a profiled event queue ddEvent : Pointer to a profiled event queue

6.2 Macro Constants

None

6.3 Functions

6.3.1 Opf_InitOsProfiler

Function Name	Opf_InitOsProfiler
Syntax	FUNC(void, OPF_CODE) Opf_InitOsProfiler(void)
Service ID	0x00
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (In)	None
Parameters (Inout)	None
Parameters (Out)	None
Return Value	None
Description	This function is called to initialize OsProfiler
Preconditions	None
Configuration Dependency	None

6.3.2 Opf_StartOsProfiler

Function Name	Opf_StartOsProfiler
Syntax	FUNC(void, OPF_CODE) Opf_StartOsProfiler(void)
Service ID	0x00
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (In)	None
Parameters (Inout)	None
Parameters (Out)	None
Return Value	None
Description	This function is called to start OsProfiler
Preconditions	None
Configuration Dependency	None

6.3.3 Opf_StopOsProfiler

Function Name	Opf_StopOsProfiler
Syntax	FUNC(void, OPF_CODE) Opf_StopOsProfiler(void)
Service ID	0x00
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (In)	None
Parameters (Inout)	None
Parameters (Out)	None
Return Value	None
Description	This function is called to stop OsProfiler
Preconditions	None
Configuration Dependency	None

6.3.4 Opf_RestartOsProfiler

Function Name	Opf_RestartOsProfiler
Syntax	FUNC(void, OPF_CODE) Opf_RestartOsProfiler(void)
Service ID	0x00
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (In)	None
Parameters (Inout)	None
Parameters (Out)	None
Return Value	None
Description	This function is called to restart OsProfiler
Preconditions	None
Configuration Dependency	None

6.3.5 Opf_GetOsProfilerRecMode

Function Name	Opf_GetOsProfilerRecMode
Syntax	FUNC(OpfRecModeType, OPF_CODE) Opf_ GetOsProfilerRecMode (void)
Service ID	0x00
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (In)	None
Parameters (Inout)	None
Parameters (Out)	None
Return Value	OpfRecModeType
Description	This function is called to get OsProfiler's record mode
Preconditions	None
Configuration Dependency	None

6.3.6 Opf_GetTaskInfo

Function Name	Opf_GetTaskInfo
Syntax	FUNC(Tdd_Opf_ProfileTaskInfo *, OPF_CODE) Opf_GetTaskInfo(uint16 LddTaskID)
Service ID	0x00
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (In)	uint16 LddTaskID
Parameters (Inout)	None
Parameters (Out)	None
Return Value	Tdd_Opf_ProfileTaskInfo *
Description	This function is called to get task information
Preconditions	None
Configuration Dependency	None

6.3.7 Opf_GetProfileInfo

Function Name	Opf_GetProfileInfo
Syntax	FUNC(Tdd_Opf_ProfileEventQ *, OPF_CODE) Opf_GetProfileInfo(void)
Service ID	0x01
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (In)	None
Parameters (Inout)	None
Parameters (Out)	None
Return Value	Tdd_Opf_ProfileEventQ *
Description	This function is called to get profile information
Preconditions	None
Configuration Dependency	None

7. Generator

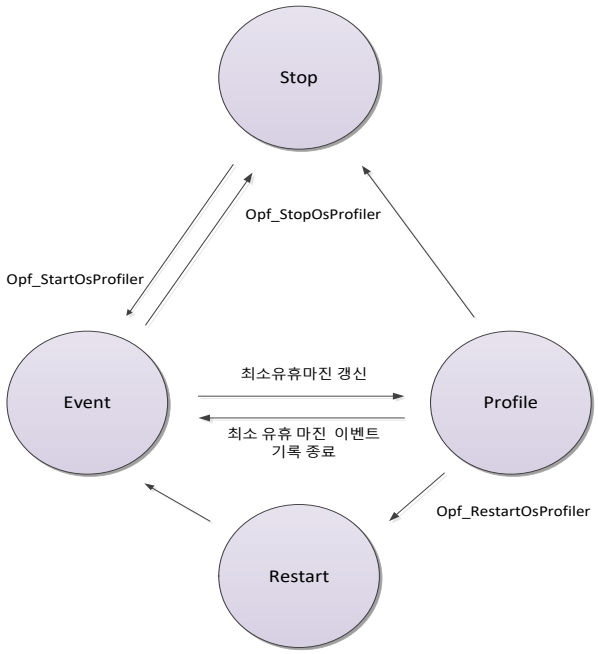
7.1 Generator Option

None

8. OS Profiler

8.1 Operational Modes

OS Profiler has four operational modes as shown in the image below. The operational mode can be identified via the API or checking a global variable in the debugger.



- When using the global variable: value stored in Opf_GddOsProfiler.ddRecMode
- When using the API: return value of Opf_GetOsProfilerRecMode()

Constant values standing for the operational modes are shown below.

Operational Modes	Constant value
OPF_RECMODE_STOP	0
OPF_RECMODE_EVENT	1
OPF_RECMODE_PROFILE	2
OPF_RECMODE_RESTART	3

8.2 System Timer Clock

It is possible to check the Clock of the System Timer that is being used by OS Profiler. This information can be obtained by using the API or a global variable on the debugger.

- When using the global variable: stored value of Opf_GddOsProfiler.ddSTMClock (in MHz)
- When using the API: contained in the Opf_GddOsProfiler struct returned by Opf_GetProfileInfo()

Some of the information collected by the OS Profiler is in the form of System Timer tick value. This value can be converted into actual time if the Clock of the System Timer is known.

8.3 Event Queue

Real-time events collected by OS Profiler can be accessed. This information can be obtained by using the API or a global variable on the debugger.

- When using the global variable: Opf_GddOsProfiler.pEventQ
- When using the API: contained in the Opf_GddOsProfiler struct returned by Opf_GetProfileInfo()

Key information stored in Opf_GddOsProfiler.ddEventQ is as follows:

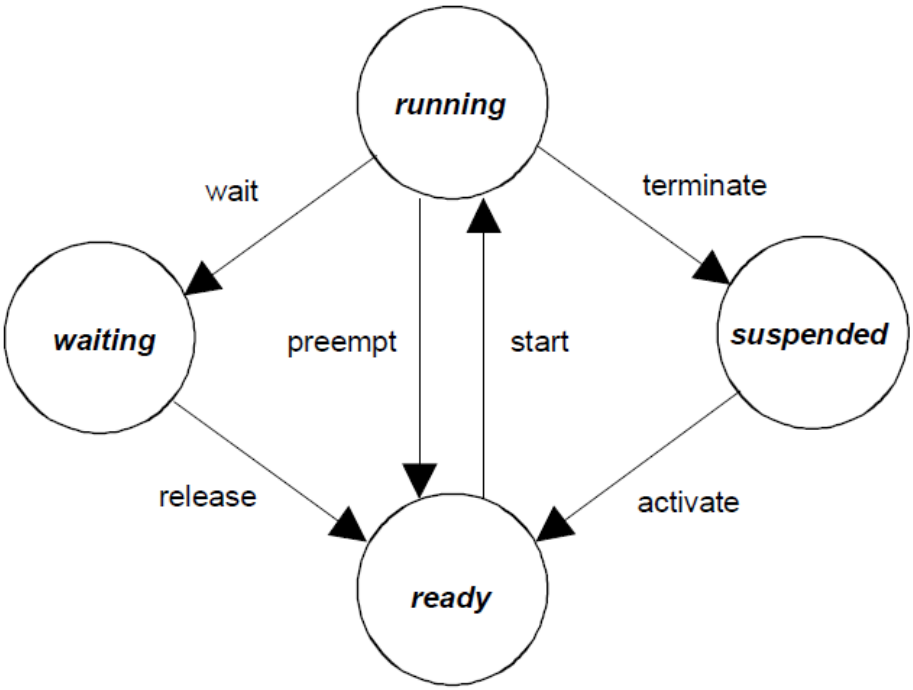
Information	Description
Opf_GddOsProfiler.pEventQ.ddEventQueueSize	The size of event queue
Opf_GddOsProfiler.pEventQ.ddLastEventPos	The position in which the last event is stored
Opf_GddOsProfiler.pEventQ.ddEventCount	The number of events occurred
Opf_GddOsProfiler.pEventQ.ddEvent	Actual event information

The actual event information stored in Opf_GddOsProfiler.pEventQ.ddEvent contains System Time (at the time when the event occurred), Time Stamp (the elapsed time since the previous event), Object Type, Object ID, and State (of the Object) as in below examples.

System Time	Time Stamp	Object Type	Object ID	State
10111	111	Task	Task A	Running
10423	312	Task	Task A	Ready
10736	313	Task	Task B	Running
10850	114	ISR	ISR 1	Entry
11265	415	ISR	ISR 1	Exit
11381	116	Task	Task A	Running
12028	647	Task	Task A	Suspended
12139	111	ISR	ISR 2	Entry
12350	211	ISR	ISR 2	Exit
12470	120	Task	Task A	Running

The types of objects collected as events by OS Profiler are limited to tasks and Category2 ISRs.
According to the OSEK OS/AUTOSAR OS standards, four task states are defined as below.

Task state	Description
Running	It means the task is occupying an MCU core and its code is actually being run. Only one task can be in this state for a single core.
Ready	It means the task is waiting to transition to the Running state. Multiple tasks can be in the Ready state at the same time; a task with the highest priority can transition to Running.
Waiting	It means the task has stopped running and waiting for a certain event. When the event being waited for occurs, the task moves to the Ready state.
Suspended	It means the task is finished.



OS Profiler divides the Ready state into Activated and Ready.

- Activated: the task is activated and waiting to switch to Running.
- Ready: the task has been preempted by another task and pushed to Ready. It will return to Running afterwards, when allowed to do so.

Category2 ISR has two states as shown below.

ISR state	Description
Enter	The Category2 ISR is about to begin execution.
Exit	The Category2 ISR has just been executed.

8.4 Profile Event Queue

The Event Queue's content at the time when the lowest waiting margin was reached is stored at the profile's Event Queue. This information can be obtained by using the API or a global variable on the debugger.

- When using the global variable: Opf_GddOsProfiler.pProfileEventQ
- When using the API: contained in the Opf_GddOsProfiler struct returned by Opf_GetProfileInfo()

Key information stored in Opf_GddOsProfiler.pProfileEventQ is as follows:

Information	Description
Opf_GddOsProfiler.pProfileEventQ.ddEventQueueSize	The size of event queue
Opf_GddOsProfiler.pProfileEventQ.ddLastEventPos	The position in which the last event is stored
Opf_GddOsProfiler.pProfileEventQ.pCurProfiledTaskInfo	Information of the task that reached the lowest waiting margin
Opf_GddOsProfiler.pProfileEventQ.ddEvent	Information of the event at the time when the lowest waiting margin was reached

The event information recorded when the lowest waiting margin was reached is stored in Opf_GddOsProfiler.pProfileEventQ.ddEvent. The format of this information is the same as the format used in the Event Queue (Opf_GddOsProfiler.pEventQ).

8.5 Slack Time

Waiting margin of a periodic task is calculated as follows (recorded in per mille).

Waiting margin of a periodic task
= ((Periodic task's activation time – periodic task's last termination time) / task period) * 1000

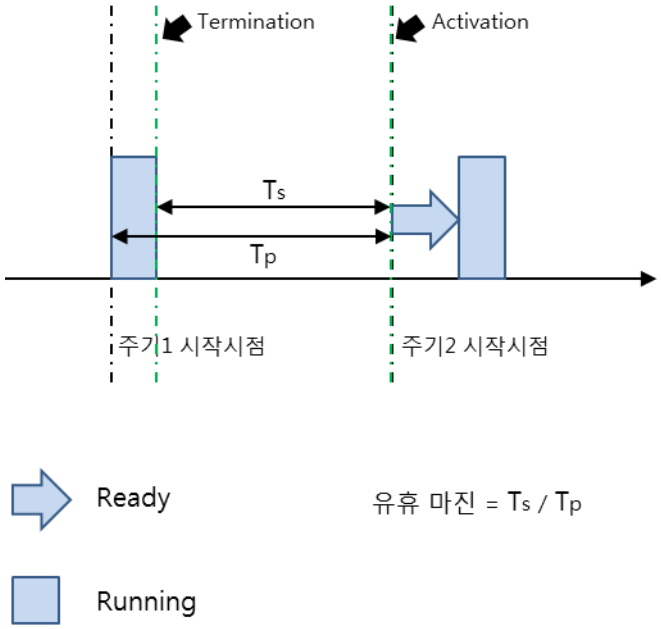


Image: Waiting margin of a periodic task

For example, as illustrated below, the waiting time of a task with a 5ms period is calculated for each period (shown using red arrows). The "waiting time" equals the periodic task's activation time minus its last termination time; the waiting time divided by the task's period is the "waiting margin" notated as a per mille value. A bigger waiting margin indicates the ability of the periodic task in question

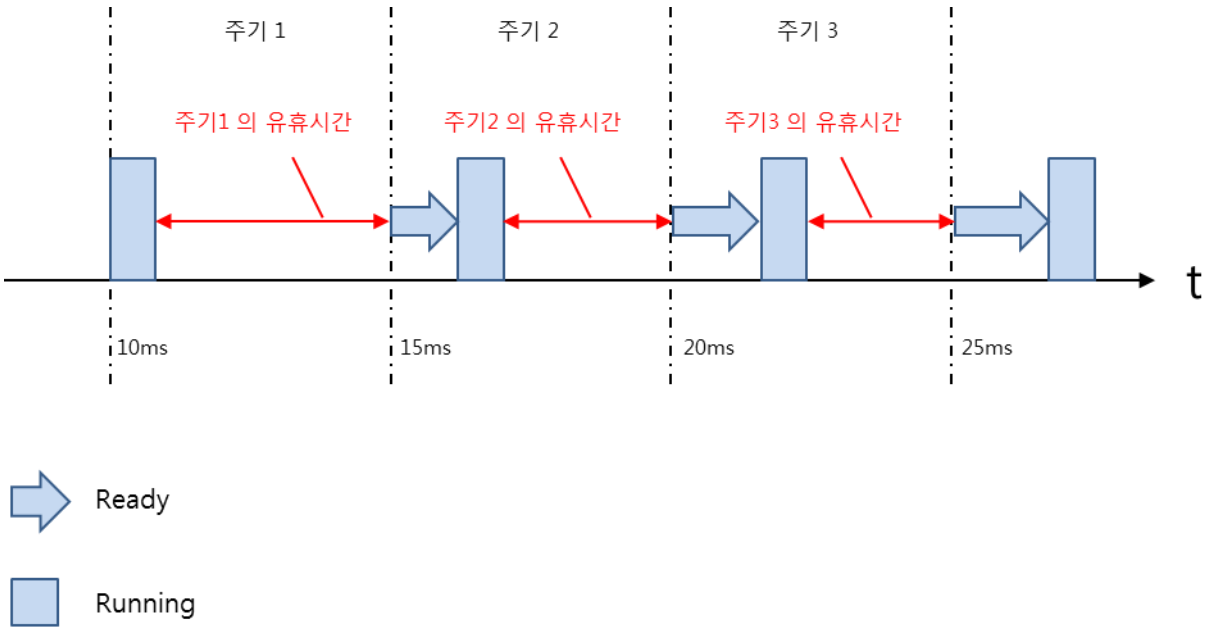


Image: Waiting margin of a periodic task

"Minimum waiting margin" refers to the smallest waiting margin of a periodic task.

Minimum Waiting Margin = MIN(waiting margin$_1$, waiting margin$_2$, waiting margin$_3$, ...)

For example, in the example below, the waiting margin of the 3rd period, among the three periods, will be the lowest waiting margin of Task A.

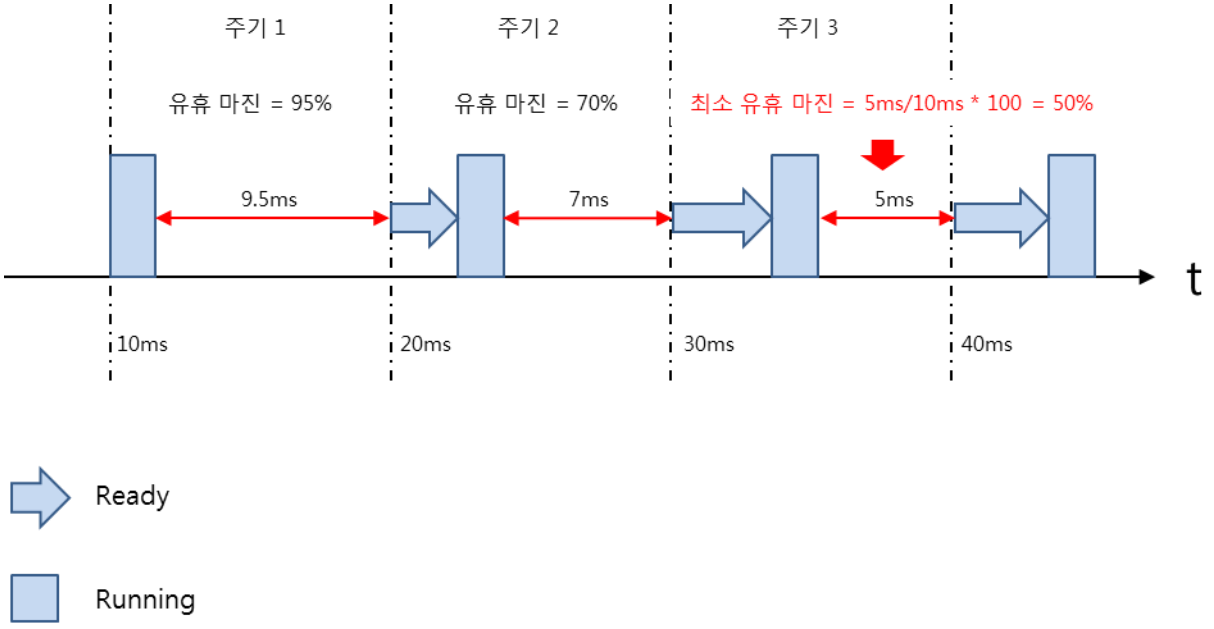


Image: lowest waiting margin.

This information can be obtained by using the API or a global variable on the debugger.

- When using the global variable: Opf_GaaProfileTaskInfo[n].ddMinST (n = 0, 1, 2, ..)
- When using the API: contained in the Opf_GaaProfileTaskInfo struct returned by Opf_GetTaskInfo()

8.6 Jitter

“Jitter” of a periodic task is calculated as follows (recorded in per mille).

Jitter of a periodic task = ((the time when the task began to run – the time when the task was activated) / task period) * 1000

For example,

Jitter of a periodic task = ((t2 – t1) / Tp) * 1000

as illustrated in the below image.

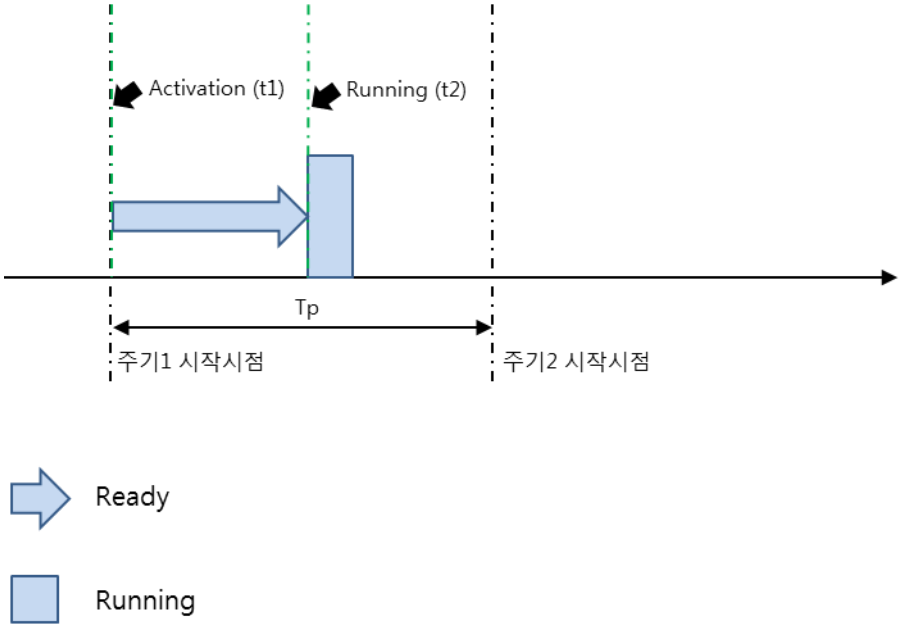


Image: jitter of a periodic task

“Max jitter” refers to the largest jitter value of a periodic task.

Max jitter of a periodic task = MAX($Jitter_1$, $Jitter_2$, $Jitter_3$, ...)

This information can be obtained by using the API or a global variable on the debugger.

- When using the global variable: Opf_GaaProfileTaskInfo[n]. ddMaxJIT (n = 0, 1, 2, ..)
- When using the API: contained in the Opf_GaaProfileTaskInfo struct returned by Opf_GetTaskInfo()

8.7 CET (Core Execution Time)

“Core execution time” refers to the pure running time of a task. Therefore, it does not include time during which the task is preempted by other tasks. However, the time during which it is preempted by ISRs are NOT excluded.

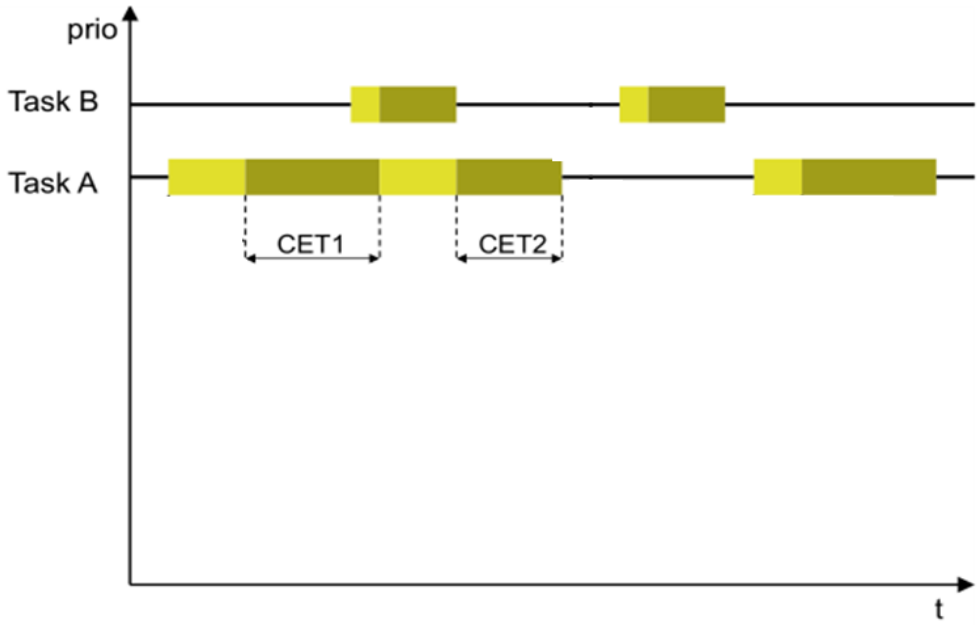


Image: CET (Core Execution Time)

"Max CET" refers to the largest CET value of a periodic task.

Max CET of a periodic task = MAX(CET_1, CET_2, CET_3, ...)

This information can be obtained by using the API or a global variable on the debugger.

- When using the global variable: Opf_GaaProfileTaskInfo[n]. ddMaxCET (n = 0, 1, 2, ..)
- When using the API: contained in the Opf_GaaProfileTaskInfo struct returned by Opf_GetTaskInfo()

8.8 GET (Gross Execution Time)

"GET (Gross Execution Time)" refers to the time from a task begins to run until it is suspended. Therefore, it does include time during which the task is preempted by other tasks.

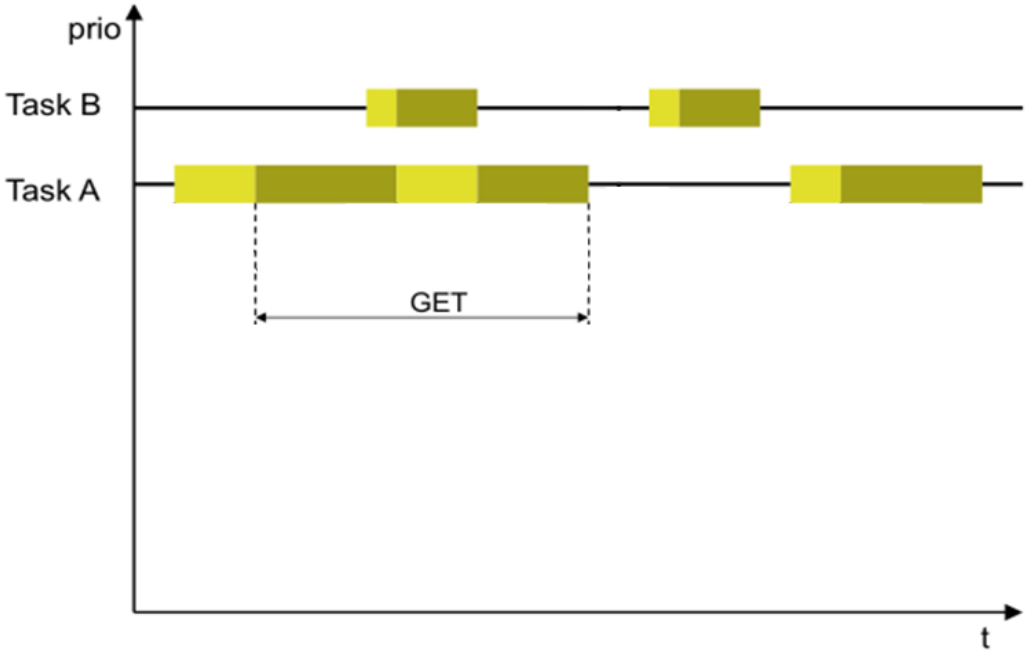


Image: GET (Gross Execution Time)

"Max GET" refers to the largest GET value of a periodic task.

Max GET of a periodic task = MAX(GET_1, GET_2, GET_3, ...)

This information can be obtained by using the API or a global variable on the debugger.

- When using the global variable: Opf_GaaProfileTaskInfo[n]. ddMaxGET (n = 0, 1, 2, ..)
- When using the API: contained in the Opf_GaaProfileTaskInfo struct returned by Opf_GetTaskInfo()

8.9 RT (Response Time)

"RT (Response Time)" refers to the time from a task is activated until it is suspended. Therefore, it does include all times during which the task is preempted by other tasks.

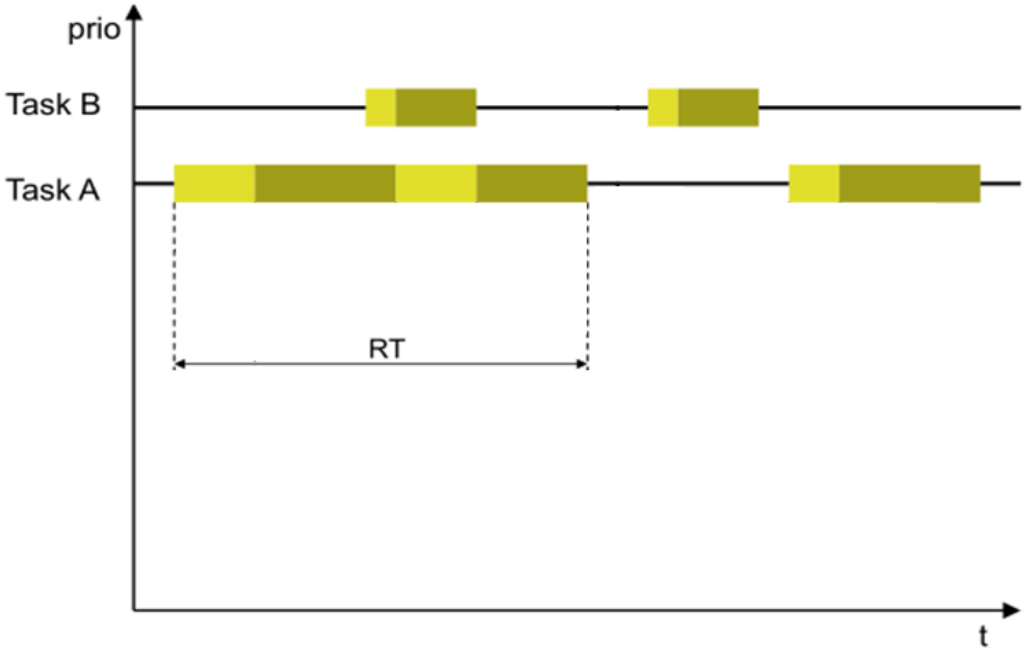


Image: RT (Response Time)

"Max RT" refers to the largest RT value of a periodic task.

Max RT of a periodic task = MAX(RT_1, RT_2, RT_3, ...)

This information can be obtained by using the API or a global variable on the debugger.

- When using the global variable: Opf_GaaProfileTaskInfo[n]. ddMaxRT (n = 0, 1, 2, ..)
- When using the API: contained in the Opf_GaaProfileTaskInfo struct returned by Opf_GetTaskInfo()

8.10 Multi-Core Support

Currently, OS Profiler supports operation on an Aurix multi-core environment. It should be noted that OS Profiler supports up to three cores on TC2xx and up to six cores on TC3xx; it assumes all cores are synchronized at the same STM frequency. OS Profiler operates the same way in a multi-core environment as on a single core with events occurring on each core being stored separately as shown below.

- Core0: Opf_GddOsProfiler
- Core#n: Opf_GddOsProfilerCore#n (#n = 1 ~ 5)

As periodic tasks run on a single core, their slack time, response time, and other variables are calculated as described above. Their values can be obtained with Opf_GaaProfileTaskInfo.

9. Appendix

9.1 Precautions on Design

None

9.2 Exclusive Areas

None