Freescale AUTOSAR OS/MPC56xxAM v.3.0 Benchmarks

User's Manual

Revised: 25 March 2010

freescale™ semiconductor

Legal Notices

Freescale reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Freescale does not assume any liability arising out of the application or use of any product described herein; neither does it convey any license under its patent rights nor the rights of others. Freescale products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale was negligent regarding the design or manufacture of the part.

The information in this document has been carefully checked and is believed to be entirely reliable, however, no responsibility is assumed for inaccuracies. Furthermore, Freescale reserves the right to make changes to any products herein to improve reliability, function or design. Freescale does not assume liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights or the rights of others.
The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement.

No part of this publication may be reproduced or transmitted in any form or by any means - graphic, electronic, electrical, mechanical, chemical, including photocopying, recording in any medium, taping, by any computer or information storage retrieval systems, etc., without prior permissions in writing from Freescale Semiconductor, Inc. Permission is granted to reproduce and transmit the Problem Report Form and the Customer Satisfaction Survey to Freescale.

Important Notice to Users

While every effort has been made to ensure the accuracy of all information in this document, Freescale assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause. Freescale further assumes no liability arising out of the application or use of any product or system described herein; nor any liability for incidental or consequential damages arising from the use of this document. Freescale disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, including implied warranties of merchantability or fitness for a particular purpose.

Contetns

1

Introduction

This manual describes a set of benchmark applications for Freescale AUTOSAR OS / MPC56xxAM v.3.0, their functionality, a technique of measurements and results. “Performance Benchmarks” chapter gives a description of the Performance Benchmarks functionality, conducted measurements and results. “Memory and Overhead Benchmarks” chapter gives a description of the benchmarks functionality used for measuring memory consumption and system overhead by some typical applications, conducted measurements and results.

This chapter consists of the following sections:

Benchmarks Overview

The Freescale AUTOSAR OS / MPC56xxAM Benchmarks is a set of applications which provides the user with possibility to evaluate performance characteristics of Freescale AUTOSAR OS / MPC56xxAM v.3.0 Benchmark package includes executable files and source code of the benchmarks.

The AUTOSAR Operating System is well documented and measured. Data for performance characteristics and memory requirements are provided in the User’s Documentation for each service. The purpose of the benchmarks is to provide the user with

impression about characteristics of some typical applications and ability to check performance on his own hardware.

The timing characteristics may be observed in two ways: reading values from the variables obtained by means of timer capture and using logic analyzer or oscilloscope to observe signals and measure timings on certain pins.

Typographical Conventions

This User's Manual employs the following typographical conventions:

Boldface type

Bold is used for important terms, notes and warnings.

Italics

Italics are used for all OSEK names of directives, macros, constants, routines and variables.

`Courier` font

The courier typeface is used for code examples in the text.

References

[1] AUTOSAR Specification of Module Operating System v.3.0.1 R.3.0 rev.0003 (AUTOSAR_SWS_OS)

[2] OSEK/VDX Operating System v.2.2.2, OSEK group, 05 July 2004

[3] Freescale AUTOSAR OS/MPC56xxAM v.3.0 Technical Reference

Abbreviations

The following acronyms and abbreviation are used in this User's Manual.

BCC	Basic Conformance Class, a defined set of functionality in OSEK, for which the waiting state of tasks is not permitted

CPU	Central Processor Unit
ECC	Extended Conformance Class, a defined set of functionality in OSEK, for which the waiting state of tasks is permitted
IDE	Integrated development environment
ISR	Interrupt Service Routine
OIL	OSEK Implementation Language
OS	Operating System
OSEK	Open systems and the corresponding interfaces for automotive electronics (in German)
RAM	Random Access Memory
ROM	Read Only Memory
SC	Scalbility Class

Setup Instructions

The executable files are built with CodeWarrioir, GreenHills and WindRiver compilers. Makefiles (for GNU make v.3.80) are provided to build benchmarks. The benchmarks were executed on MPC5644A MCU with Lauterbach debugger, each benchmark directory has an appropriate .cmm file for loading benchmark executable. The structure of benchmarks directories is stated below:

- `MAN` - Benchmark User's Manual
- `COMMON` - common files shared by all benchmarks
- `PERFORMANCE` - benchmarks for performance measurements
  - `COMMON` - source code for performance benchmarks
  - `BCC1` - application for the BCC1 conformance class
  - `ECC1` - application for the ECC1 conformance class
- `MEMORY` - benchmarks for memory and system overhead measurements
  - `COMMON` - source code for memory benchmarks
  - `BCC1` - application for the BCC1 conformance class
  - `ECC1` - application for the ECC1 conformance class

Under each BCC1, ECC1 directories there are four subdirectories for benchmark applications configured for Classes SC1 .. SC4.

Each benchmark application consists of the application source files, an OIL file describing the OSEK configuration used for the application, executable files and a protocol of the link process - the map file, makefile used for building the benchmark application and Lauterbach command file (.cmm) for loading the executable into the debugger.

The naming convention for the benchmarks files is defined and follows the rules listed here:

- benchmark identifier consists of the following parts:
  - CC - Conformance Class: `bcc1` or ecc1
  - type of benchmark: `m` for body benchmarks, `me` for engine benchmark, `p` for performance benchmark
  - SC - Scalability Class: sc1 .. sc4

Each benchmark has the following structure:

- files located in the root directory:
  - makefile and Lauterbach script.
- `sources` directory contains benchmark application OIL file
- `bin` contains the executable files of the benchmarks built by the supported compilers in `ELF` format and `MAP` files
- additionally to the listed above files common files shared by all benchmarks are contained in the common directory.

Benchmarks are developed to work on the XPC56XX EVB board:

Benchmark Measurement Considerations

Measurements of execution time may be performed by means of timer capture at particular points of the execution flow and by means of output port pin manipulations at the same points. Executables included in the benchmark installation only provide the timer capture method of time measurements. The STM timer counter is used for timing measurements. GPIO pins are used for output. Timer capture and / or pin manipulations take some time of the program execution thus they slightly increase the result timings.

More accurate data can be obtained by switching off one of the measurement methods when another one is being used. There are two defines “USEPORT” and “USETIMER” at the beginning of the common header files “benchmark.h” in the common directory of each performance benchmark. Comment the line with “USETIMER” and uncomment the line with “USEPORT” to work with an oscilloscope and / or a logic analyzer, then rebuild a benchmark. Provided benchmark executables are built with undefined macro “USEPORT” and data available in this manual are measured within this configuration.All measurements for OS / MPC56xx are performed at 64 MHz clock frequency. The mesurements were performed with following MCU setings:

- cache disabled;
- BTB (branch prediction) enabled;
- executable code is loaded into SRAM.

The estimated accuracy of measurements is about ±4 cycles (±0.066µs).

Building Benchmark Executables

Benchmark executables may be rebuilt by the user with other configuration options or to adopt them to some particular hardware. Freescale AUTOSAR OS / MPC56xxAM v.3.0 shall be installed on a computer to rebuild benchmarks.

To changecompiler please edit the correspondent makefile to set a compiler and then run GNU make.

Technical Support Information

How to Contact Us

Corporate Headquarters

```
Freescale Semiconductor, Inc.
7700 West Parmer Lane
Austin, TX 78729
U.S.A.
```

Technical Support http://www.freescale.com/support

2

Performance Benchmarks

This chapter of the manual describes benchmark applications for performance measurement. The applications are configured for BCC1 and ECC1 OSEK Conformance Classes and for all AUTOSAR Scalability Classes (SC1 - SC4). Configuration, execution sequence and performance measurements are introduced in detail for each benchmark application.

Note, that for this RTM release benchmark measurements were done with CPU cache switched off

This chapter includes the following sections:

Benchmark Application for BCC1

The benchmark application for the BCC1 allows the following performance characteristics measurements:

- Switch time from one task to another
- Switch time from ISR to a task
- ISR entry time
- Time required to serve alarm(s)
- Time required to serve Schedule Table

All these measurements are performed within one application. The files of this benchmark are located in the `PERFORMANCE\BCC1\SCn` directories.

- Open the file benchmark\performance\common\test.cmm and change the line with <&suff="_gh"> - set it “_cw” for CodeWarrior or “_db” for WindRiver if required.

- Connect Logic analyzer or oscilloscope (if any) to the Evaluation Board GPIO pins. For details of the pin usage please look into the description of benchmark functionality below
- Start the debugger
- Open `PERFORMANCE\BCC1\SCn\pbcc1sc<n>.cmm` debugger configuration file via menu “File -> Run Batch file”
- Wait until the benchmark executable is loaded, then starts execution and then stops.
- read results in the log window.

Configuration

The following OS configuration is used for the benchmark application:

- STANDARD OS status
- DEBUG_LEVEL equals to 0
- No hooks are defined
- Full preemptive scheduler
- ResourceScheduler support is off
- FastTerminate is set to TRUE
- FastScheduler is set to TRUE

Time is calculated for CPU Clock frequency 64 MHz.

The same hardware configuration is used for all benchmarks measurements, see “Setup Instructions”.

The benchmark application consists of the following OSEK OS objects:

- OneTaskISR (Isr1) - an interrupt handler for the eMIOS channel 16 timer
- SystemTimerISR[1] - a system timer interrupt handler for the eMIOS channel 8 interrupt, it is configured to generate interrupt

[1] The application code is used for processing the system and second timer’s ISRs instead of the ones provided by the OS. The changes in an ISR’s code were made to incorporate the code for the measurements.

every 0.256 ms. The Schedule Table is attached to the system timer and the timer is activated by the StartSchedule Table service call

- SecondTimerISR - the second system timer interrupt handler working from the PIT channel 0 timer interrupts, it is configured to generate interrupt every 0.256 ms. The set of OSEK alarms is attached to this timer
- TASK_1 - a task activated from ISR, serves to measure switch time from ISR to a task and between tasks
- TASK_2 - a task activated from TASK_1, serves to measure switch time between tasks
- TASK_ALM - a task activated by an alarm, serves to measure switch time from ISR to a task with alarm expiration
- TASK_t1..t7 - tasks activated by the Schedule Table on the corresponding step with the 1.024 ms period. The tasks attached to the Schedule Table have in their names a t1-t7 postfix with the number corresponding to the step number. Note that steps 4-7 have zero time between them and tasks 4-7 are activated at the same timer tick
- TASK_h1..h8 - a set of tasks called “harmonic” because they all have harmonic periods and at some moments either 4 or 8 tasks may be activated; TASK_h1 has the highest priority in this set of tasks and the 0.256 ms period, it is activated in each interrupt. Tasks 2-4 have the 0.512 ms period, tasks 5-8 have 1.024 ms period; the bigger the task number of tasks h1...h8 is the lower priority this task has
- TASK_BGND - a background task with the lowest priority, performs initialization of the measurements sequences

The timing measurements are performed by means of counting the number of Time Basecycles made by reading counter; and the results are shown in tables in the columns titled “cycles” and after recalculation in microseconds for 64 MHz clock they are placed in the columns titled “µs”.

GPIO is used by a benchmark application to output signals from ISRs and tasks. These signals mark the actions taken, which allows observing the performance numbers using the Logic analyzer or oscilloscope. In order to use this feature one has to recompile the benchmark application, see also “Building Benchmark Executables”.

The usage of the pins by the ISRs and tasks is indicated in Table 2.1. The pin numbers correspond to the pin numbers of Port eTPUA, i.e. PIN0 is eTPUA0 .

Table 2.1 Pin Action Description

ISRs/Tasks	Priority	Pin	at beg.	at end	Period
OneTaskISR	5	PIN4	set[a]	reset	one-shot
SystemTimerISR	2	PIN0	set	reset	0.256 ms
SecondTimerISR	7	PIN4	set	reset	0.256 ms
TASK_1	21	PIN4	set	-	one-shot
TASK_2	22	PIN5	set	-	one-shot
TASK_ALM	23	PIN5	set	-	one-shot
TASK_t1	11	PIN0	set	-	1.024 ms
TASK_t2	12	PIN1	set	-	1.024 ms
TASK_t3	13	PIN2	set	-	1.024 ms
TASK_t4	14	PIN0	set	-	1.024 ms
TASK_t5	15	PIN1	set	-	1.024 ms
TASK_t6	16	PIN2	set	-	1.024 ms
TASK_t7	17	PIN3	set	-	1.024 ms
TASK_h1	8	PIN4	set	-	0.256 ms
TASK_h2	7	-	-	-	0.512 ms
TASK_h3	6	-	-	-	0.512 ms
TASK_h4	5	PIN5	set	-	0.512 ms
TASK_h5	4	-	-	-	1.024 ms
TASK_h6	3	-	-	-	1.024 ms
TASK_h7	2	-	-	-	1.024 ms
TASK_h8	1	PIN6	set	-	1.024 ms
TASK_BGND	0	PIN7	toggle		N/A

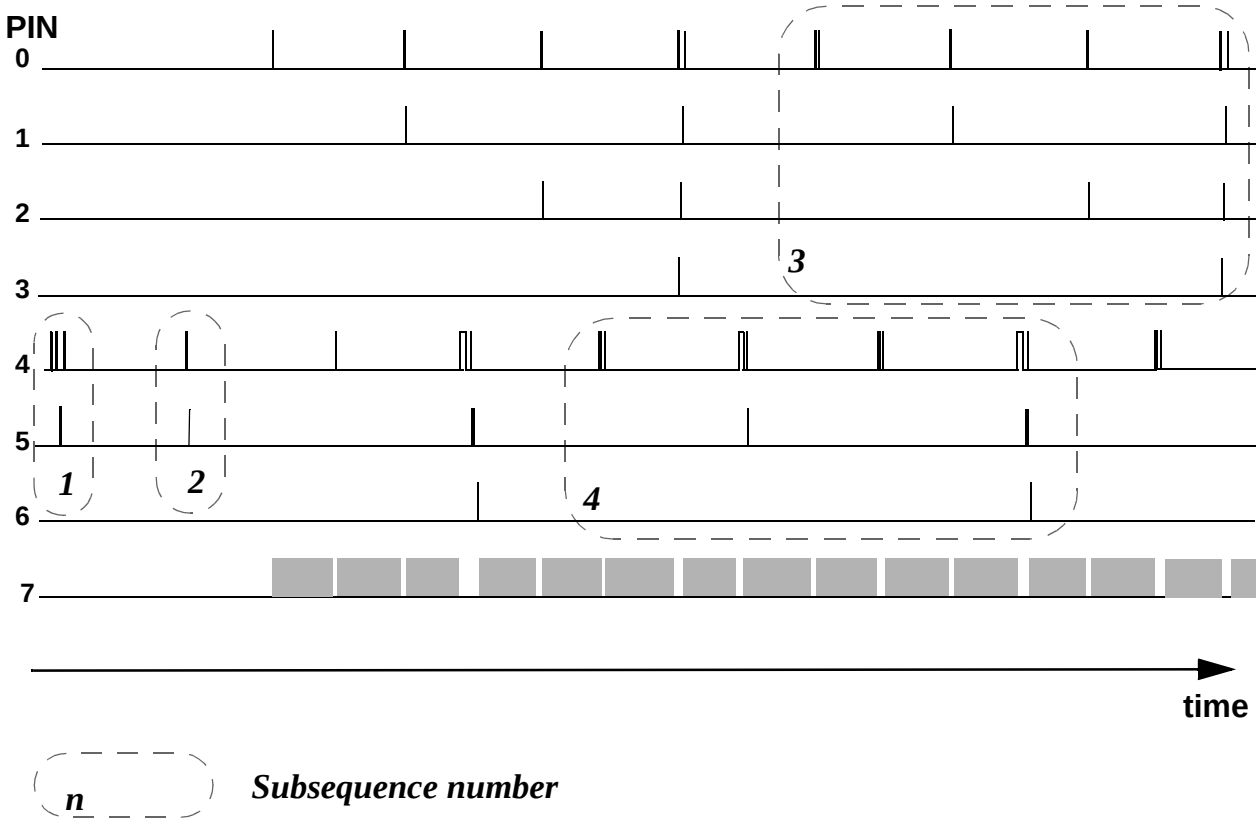
a. When any pin is set, all other are cleared (reset).

Execution Sequence

TASK_BGND is autostarted and performs initialization of measurement sequence.

Figure 2.1 shows the execution sequence for an application with pin states. The subsequences are marked with borders and the detailed descriptions of each measurement can be found below.

Figure 2.1 Execution Sequence for BCC1 Performance Benchmark



Subsequence to Measure Tasks Activate/Terminate Timings

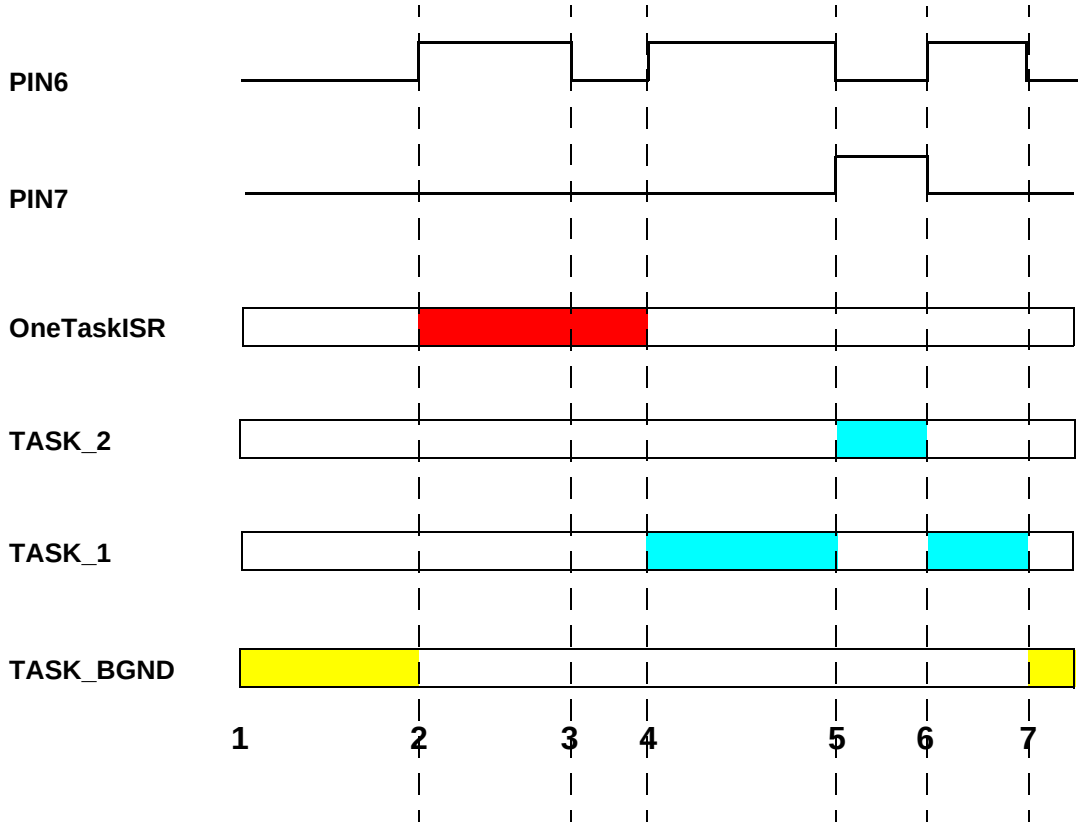
Measurements: ISR entry latency, switch time from ISR to tasks and between tasks.

1. TASK_BGND initializes timer in order to raise an interrupt, processed by the OneTaskISR, in ms and then this task waits in loop until the flag variable has been set by the TASK_2
2. OneTaskISR activates TASK_1
3. OneTaskISR exits

4. TASK_1 activates TASK_2
5. TASK_2 starts and terminates
6. After TASK_2 terminates the control returns to TASK_1 and TASK_1 terminates
7. After TASK_1 terminates the control returns to TASK_BGND

Figure 2.2 shows the execution subsequence for measuring tasks activate / terminate timings.

Figure 2.2 Task Activation/Termination Time Measurement



Task and ISR Activation/Termination times for CodeWarrior, WindRiver and GreenHills are shown in the tables below.

Table 2.2 Task Activation/Termination Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Interrupt entry latency	- [a]	133	2,08	286	4,47	221	3,45	324	5,06
ActivateTask from ISR	2-3	97	1,52	87	1,36	130	2,03	132	2,06
From ISR to TASK_1	3-4	186	2,91	336	5,25	319	4,98	423	6,61
Activate to TASK_2	4-5	161	2,52	313	4,89	307	4,80	430	6,72
Terminate to TASK_1	5-6	89	1,39	296	4,63	256	4,00	349	5,45
Return to background	6-7	160	2,50	301	4,70	324	5,06	394	6,16

a. The interrupt entry latency is measured from time of interrupt request from hardware raised to the first command of user ISR.

Table 2.3 Task Activation/Termination Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Interrupt entry latency	- [a]	164	2,56	370	5,78	223	3,48	365	5,70
ActivateTask from ISR	2-3	98	1,53	105	1,64	165	2,58	156	2,44
From ISR to TASK_1	3-4	163	2,55	374	5,84	333	5,20	450	7,03
Activate to TASK_2	4-5	120	1,88	396	6,19	331	5,17	501	7,83
Terminate to TASK_1	5-6	72	1,13	299	4,67	245	3,83	352	5,50
Return to background	6-7	122	1,91	281	4,39	284	4,44	360	5,63

a. The interrupt entry latency is measured from time of interrupt request from hardware raised to the first command of user ISR.

Table 2.4 Task Activation/Termination Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Interrupt entry latency	- [a]	131	2,05	293	4,58	205	3,20	310	4,84
ActivateTask from ISR	2-3	73	1,14	92	1,44	126	1,97	123	1,92
From ISR to TASK_1	3-4	135	2,11	277	4,33	287	4,48	345	5,39
Activate to TASK_2	4-5	102	1,59	259	4,05	266	4,16	354	5,53

Table 2.4 Task Activation/Termination Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Terminate to TASK_1	5-6	66	1,03	214	3,34	214	3,34	264	4,13
Return to background	6-7	130	2,03	232	3,63	278	4,34	303	4,73

[a.] The interrupt entry latency is measured from time of interrupt request from hardware raised to the first command of user ISR.

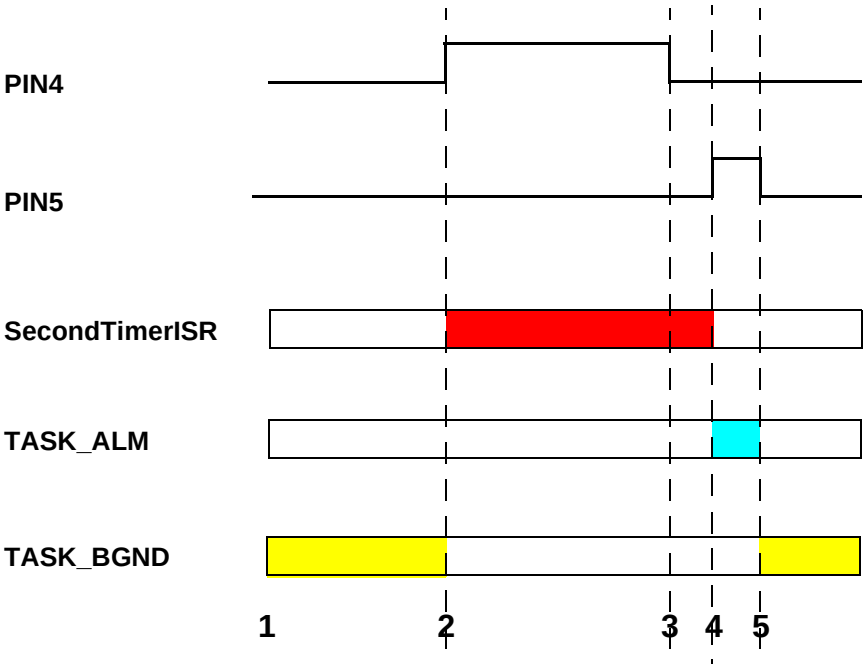
Subsequence to Measure Alarm Timings

Measurements: Task activation by alarm expiration, switch time from ISR to task.

1. TASK_BGND sets one short alarm ALARM_1 on Counter2 attached to the SecondTimerISR and then waits in loop until the flag variable has been set by the TASK_ALM
2. SecondTimerISR performs CounterTrigger service, the alarm expires and the task is activated by the alarm
3. SecondTimerISR exits
4. TASK_ALM starts and terminates
5. After TASK_ALM terminates, the control returns to TASK_BGND

Figure 2.3 shows the execution subsequence for measuring the alarm timings.

Figure 2.3 Task Activation by Alarm Time Measurement



The tasks activation by alarm expiration measurement results for CodeWarrior, WindRiver and GreenHills are indicated in the tables below.

Table 2.5 Task Activation by Alarm Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 alarm - Activate via Alarm	2-3	341	5,33	456	7,13	444	6,94	501	7,83
From ISR to TASK_ALM	3-4	178	2,78	328	5,13	311	4,86	412	6,44
Return to background	4-5	160	2,50	301	4,70	324	5,06	394	6,16

Table 2.6 Task Activation by Alarm Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 alarm - Activate via Alarm	2-3	343	5,36	502	7,84	401	6,27	490	7,66

Table 2.6 Task Activation by Alarm Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
From ISR to TASK_ALM	3-4	157	2,45	371	5,80	330	5,16	447	6,98
Return to background	4-5	122	1,91	281	4,39	284	4,44	360	5,63

Table 2.7 Task Activation by Alarm Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 alarm - Activate via Alarm	2-3	291	4,55	428	6,69	383	5,98	439	6,86
From ISR to TASK_ALM	3-4	129	2,02	272	4,25	282	4,41	342	5,34
Return to background	4-5	130	2,03	232	3,63	278	4,34	303	4,73

Subsequences to Measure Schedule Table and “Harmonic Alarms” Timings

This is the last subsequence and it is cycled. Schedule Table and Harmonic Alarms measurements do not interfere because they occur at different moments.

TASK_BGND sets 8 alarms for tasks TASK_h[1...8] and initializes Schedule Table at such a moment that interrupts for the system and the second timers occur with the ms interval. These two measurements will be described separately.

Schedule Table Performance Measurement

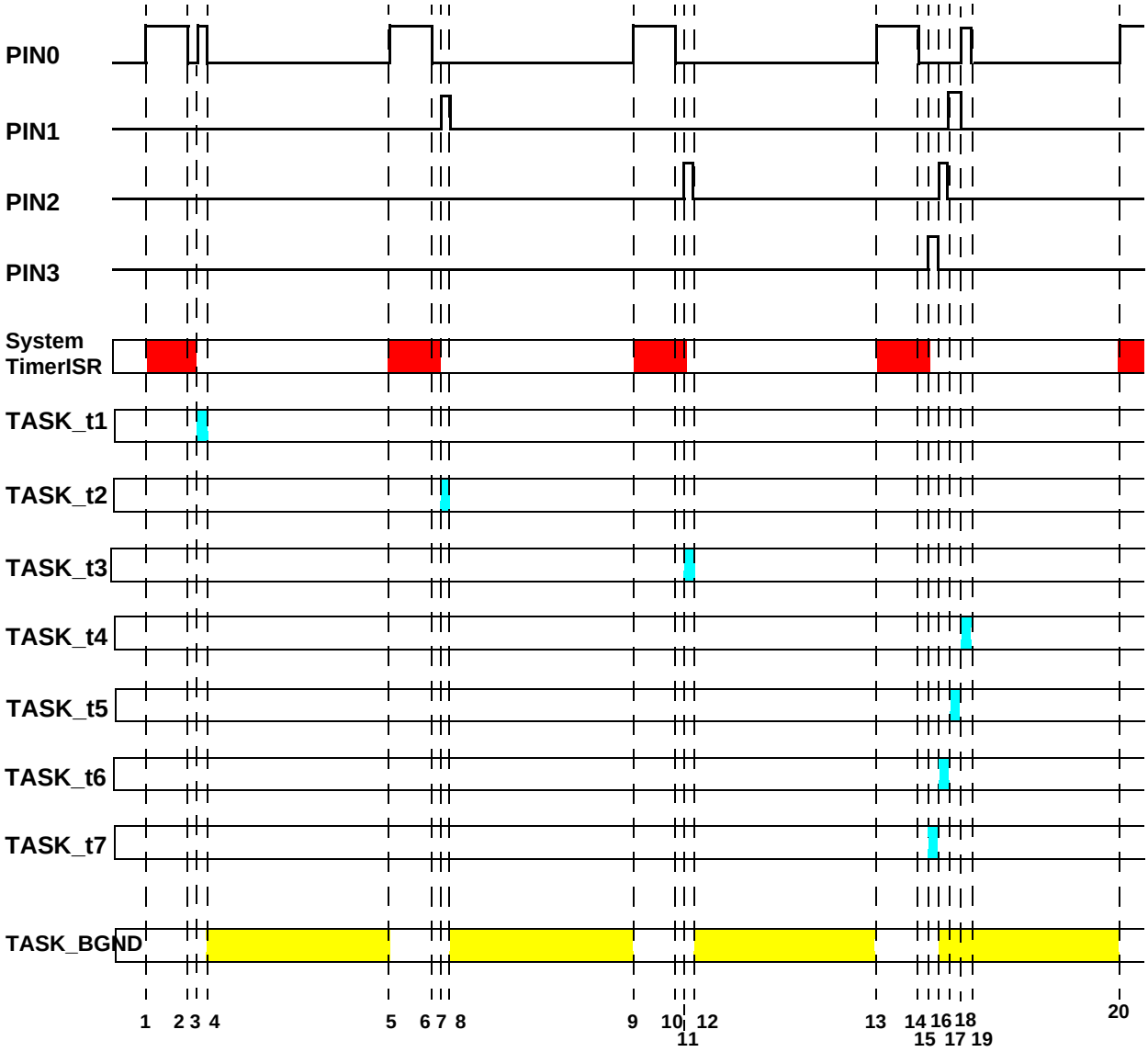
At points 1, 5, 9, 13, 17 the interrupt of SystemTimerISR occurs and the task corresponding to the appropriate step of the Schedule Table is activated. Point 1 corresponds to step 1, point 5 corresponds to step 2, point 9 corresponds to step 3, point 13 corresponds to step 4 and point 17 also corresponds to step 1. Every time the SystemTimerISR performs the time scale processing before next point.

Points 2, 6, 10, 14 correspond to the end of the ISR code execution.

Every task activated by the Schedule Table only calls the TerminateTask and returns the control to the TASK_BGND.

Figure 2.4 shows the execution subsequence for measuring the Schedule Table performance. The interval between adjacent system timer interrupts is ms, but the scale is unproportional because some of the measured intervals are insignificant compared to the time between interrupts.

Figure 2.4 Schedule Table Time Measurement



The Schedule Table performance measurement results for CodeWarrior, WindRiver and GreenHills are indicated in the tables below.

Table 2.8 Schedule Table Time Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Schedule Table ISR - 1 task activated	1-2, 5-6, 9-10	486	7,59	548	8,56	525	8,20	577	7.63
Schedule Table ISR - 4 tasks activated	13-14	667	10,42	746	11,66	733	11,45	783	9.88
From ISR to TASK_t1	2-3	162	2,53	302	4,72	286	4,47	372	4.44
From ISR to TASK_t2	6-7	162	2,53	302	4,72	286	4,47	372	4.33
From ISR to TASK_t3	10-11	162	2,53	302	4,72	286	4,47	372	4.44
From ISR to TASK_t7	14-15	162	2,53	302	4,72	286	4,47	372	4.45
TASK_t7 to TASK_t4[a]	15-18	181	2,83	709	11,08	642	10,03	922	11.08
Return to background	3-4, 7-8, 11-12, 18-19	160	2,50	301	4,70	324	5,06	394	6,16

a. Switch time between individual tasks are not measured

Table 2.9 Schedule Table Time Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Schedule Table ISR - 1 task activated	1-2, 5-6, 9-10	448	7,00	522	8,16	448	7,00	499	7,80
Schedule Table ISR - 4 tasks activated	13-14	645	10,08	736	11,50	651	10,17	708	11,06
From ISR to TASK_t1	2-3	120	1,88	314	4,91	290	4,53	386	6,03
From ISR to TASK_t2	6-7	122	1,91	312	4,88	288	4,50	388	6,06
From ISR to TASK_t3	10-11	120	1,88	314	4,91	290	4,53	386	6,03
From ISR to TASK_t7	14-15	122	1,91	315	4,92	291	4,55	388	6,06
TASK_t7 to TASK_t4[a]	15-18	189	2,95	730	11,41	642	10,03	987	15,42
Return to background	3-4, 7-8, 11-12, 18-19	122	1,91	281	4,39	284	4,44	360	5,63

a. Switch time between individual tasks are not measured

Table 2.10 Schedule Table Time Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Schedule Table ISR - 1 task activated	1-2, 5-6, 9-10	407	6,36	495	7,73	441	6,89	486	7,59
Schedule Table ISR - 4 tasks activated	13-14	570	8,91	692	10,81	637	9,95	680	10,63
From ISR to TASK_t1	2-3	112	1,75	245	3,83	245	3,83	306	4,78
From ISR to TASK_t2	6-7	112	1,75	245	3,83	245	3,83	306	4,78
From ISR to TASK_t3	10-11	112	1,75	245	3,83	245	3,83	306	4,78
From ISR to TASK_t7	14-15	112	1,75	245	3,83	245	3,83	306	4,78
TASK_t7 to TASK_t4[a]	15-18	187	2,92	497	7,77	527	8,23	687	10,73
Return to background	3-4, 7-8, 11-12, 18-19	130	2,03	232	3,63	278	4,34	303	4,73

[a.] Switch time between individual tasks are not measured

"Harmonic Alarms" Performance Measurement

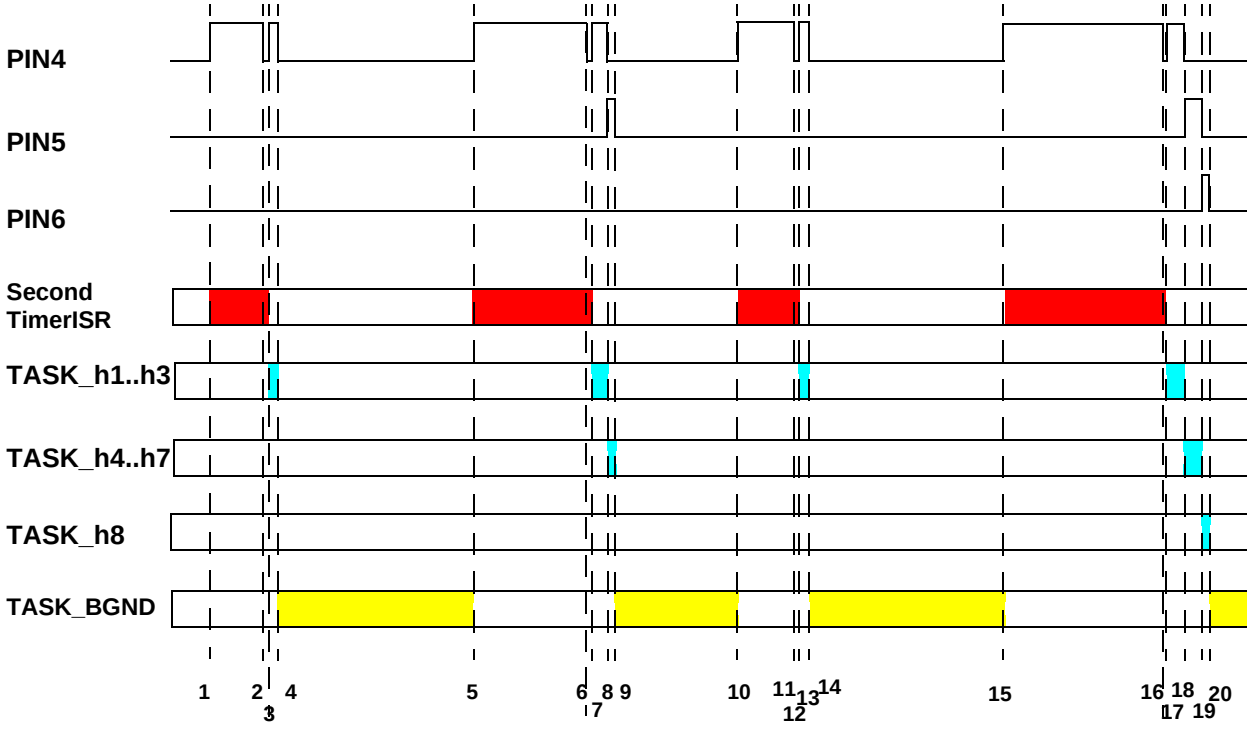
At points 1, 5, 10, 15 the interrupt of the SecondTimerISR occurs and the alarms attached to the Counter2 are processed. At points 1, 10 one alarm expires and TASK_h1 is activated. At point 2 four alarms expire and TASK_h1..h4 are activated. At point 15 eight alarms expire and TASK_h1..h8 are activated. Every time the SecondTimerISR performs counter and alarms processing before the next point.

Points 2, 6, 11, 16 correspond to the end of the ISR code execution.

Every task activated by alarms only calls the TerminateTask and return the control to the TASK_BGND.

Figure 2.5 shows the execution subsequence for measuring the "Harmonic alarms" performance. The interval between adjacent system timer interrupts is ms, but the scale is unproportional because some of the measured intervals are insignificant compared to the time between interrupts.

Figure 2.5 “Harmonic Alarms” Time Measurement



The “Harmonic alarms” performance measurement results for CodeWarrior, WindRiver and GreenHills are indicated in the tables below.

Table 2.11 “Harmonic alarms” Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 alarm[a]	1-2, 10-11	430	6,72	544	8,50	557	8,70	615	9,61
SecondTimerISR: 4 alarms	5-6	533	8,33	668	10,44	686	10,72	746	11,66
SecondTimerISR: 8 alarms	15-16	668	10,44	833	13,02	855	13,36	913	14,27
From ISR to TASK_h1	2-3, 6-7, 11-12, 16-17	178	2,78	328	5,13	311	4,86	412	6,44
TASK_h1 to TASK_h4	7-8,17-18	178	2,78	709	11,08	644	10,06	922	14,41
TASK_h4 to TASK_h8	18-19	237	3,70	955	14,92	857	13,39	1232	19,25

a. The time for the counter processing depends on the number of the alarms actually set. Compare to timing for the SecondTimerISR in Table 2.5 when only one alarm is set.

Table 2.12 "Harmonic alarms" Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 alarm[a]	1-2, 10-11	444	6,94	602	9,41	520	8,13	593	9,27
SecondTimerISR: 4 alarms	5-6	617	9,64	787	12,30	708	11,06	789	12,33
SecondTimerISR: 8 alarms	15-16	836	13,06	1022	15,97	959	14,98	1044	16,31
From ISR to TASK_h1	2-3, 6-7, 11-12, 16-17	158	2,47	372	5,81	331	5,17	448	7,00
TASK_h1 to TASK_h4	7-8,17-18	189	2,95	732	11,44	644	10,06	986	15,41
TASK_h4 to TASK_h8	18-19	240	3,75	985	15,39	861	13,45	1317	20,58

a. The time for the counter processing depends on the number of the alarms actually set. Compare to timing for the SecondTimerISR in Table 2.6 when only one alarm is set.

Table 2.13 "Harmonic alarms" Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 alarm[a]	1-2, 10-11	388	6,06	538	8,41	516	8,06	551	8,61
SecondTimerISR: 4 alarms	5-6	535	8,36	717	11,20	685	10,70	725	11,33
SecondTimerISR: 8 alarms	15-16	721	11,27	950	14,84	912	14,25	950	14,84
From ISR to TASK_h1	2-3, 6-7, 11-12, 16-17	130	2,03	271	4,23	281	4,39	340	5,31
TASK_h1 to TASK_h4	7-8,17-18	186	2,91	498	7,78	528	8,25	689	10,77
TASK_h4 to TASK_h8	18-19	242	3,78	669	10,45	703	10,98	919	14,36

a. The time for the counter processing depends on the number of the alarms actually set. Compare to timing for the SecondTimerISR in Table 2.7 when only one alarm is set.

Benchmark Application for ECC1

The benchmark application for the ECC1 is similar to the benchmark application for the BCC1 with an additional measurement related to Extended tasks.

The same measurements that were fulfilled within the BCC1 benchmark application are provided by this application but both methods – task activation and event setting - are used for task switching.

All these measurements are performed in one application. The files of this benchmark are located in a `PERFORMANCE\ECC1` directory.

To execute the benchmark application in a Lauterbach debugger take the following actions (change <n> to 1..4 according to SC):

- Open the file bencmark\performance\common\test.cmm and change the line with <&suff="_gh"> - set it “_cw” for CodeWarrior or “_db” for WindRiver if required.
- Connect Logic analyzer or oscilloscope (if any) to the Evaluation Board GPIO pins. For details of the pin usage please look into the description of benchmark functionality below
- Start the debugger
- Open `MEMORY\ECC1\SCn\pecc1sc<n>.cmm` debugger configuration file via menu “File -> Run Batch file”
- Wait until the benchmark executable is loaded, then starts execution and then stops.
- read results in the log window.

Configuration

The following OS configuration is used for the benchmark application:

- STANDARD OS status
- DEBUG_LEVEL equals to 0
- No hooks are defined
- Full preemptive scheduler
- Both Basic and Extended tasks are defined

- ResourceScheduler support disabled

Time is calculated for CPU Clock frequency 64 MHz

The same hardware configuration is used for all benchmarks measurements, see "Setup Instructions".

The benchmark application consists of the following OSEK OS objects:

- SystemTimerISR[1] - a system timer interrupt handler working from the interrupt, it is configured to generate interrupt every 0.256 ms. The Schedule Table is attached to the system timer and the timer is started by the StartSchedule Table service call
- SecondTimerISR - the second system timer interrupt handler working from the PIT channel 0 timer interrupts, it is configured to generate interrupt every 0.256 ms. A set of OSEK alarms is attached to this timer
- OneTaskISR (Isr1) - the interrupt handler working from the eMIOS timer interrupts, it is enabled from the background task to occur once and is used to activate a task
- OneEventISR (Isr2) - the interrupt handler working from the eMIOS channel 23 timer interrupts, it is enabled from the background task to occur once and is used to set an event for the waiting task
- TASK_1 - a task activated from ISR, serves to measure switch time from ISR to a task and between tasks
- TASK_2 - a task activated from TASK_1, serves to measure switch time between tasks
- TASK_3_EVT - a task autostarted and switched to waiting state, then resumed from OneEventISR, serves to measure switch time from ISR to a task after event setting
- TASK_4_EVT - a task autostarted and switched to waiting state, then resumed from TASK_3_EVT, serves to measure switch time between extended tasks on event setting
- TASK_ALM - a task activated by an alarm, serves to measure switch time from ISR to task with alarm expiration

[1] The application code is used for processing the system and the second timer's ISRs instead of the ones provided by the OS. Changes in an ISR's code were made to incorporate the code for the measurements.

- TASK_ALM_EVT - a task resumed by an alarm, serves to measure switch time from ISR to task with alarm expiration with event setting
- TASK_t1..t7 - tasks activated by the Schedule Table on the corresponding step with the 1.024 ms period. The tasks attached to the Schedule Table have in their names a t1-t7 postfix with the number corresponding to a step number. Note that steps 4-7 have zero time between them and tasks 4-7 are activated at the same timer tick
- TASK_h1..h8 - a set of tasks called “harmonic” because they all have harmonic periods and at some moments either 1 or 4 tasks may be notified, tasks 1-4 are extended tasks and they are notified by event setting, tasks 5-8 are basic tasks and they are notified by task activation, TASK_h1 has the highest priority in the set of extended tasks and TASK_h5 has the highest priority in the set of basic tasks, they both have the 0.512 ms period, but they are notified while different interrupt occurrences with the 0.256 ms interval. Tasks h2-h4 and h6-h8 have 1.024 ms period. The bigger the task number of tasks h1...h8 is the lower priority this task has
- TASK_BGND - a background task with the lowest priority, performs initialization of the measurements sequences

The timing measurements are performed by means of counting the number of Time Base cycles made by reading its counter; and the results are shown in the tables in the columns titled “cycles” and after recalculation in microseconds for 64 MHz clock they are placed in the columns titled “µs”.

The GPIO is used by a benchmark application to output signals from ISRs and tasks. These signals mark the actions taken, which allows observing the performance numbers using the Logic analyzer or oscilloscope. In order to use this feature one has to recompile the benchmark application, see also “Building Benchmark Executables”.

The usage of the pins by the ISRs and tasks is indicated in Table 2.14. The pin numbers correspond to the pin numbers of Port F, i.e. PIN0 is PF0..

Table 2.14 Pin Action Description

ISRs/Tasks	Priority	Pin	at beg.	at end	Period
OneTaskISR	5	PIN4	set[a]	reset	one-shot
OneEventISR	6	PIN4	set	reset	one-shot
SystemTimerISR	2	PIN0	set	reset	0.256 ms
SecondTimerISR	8	PIN4	set	reset	0.256 ms
TASK_1	21/basic	PIN4	set	-	one-shot
TASK_2	22/basic	PIN5	set	-	one-shot
TASK_ALM	23/basic	PIN5	set	-	one-shot
TASK_3_EVT	24/ext	PIN4	set	-	one-shot
TASK_4_EVT	25/ext	PIN5	set	-	one-shot
TASK_ALM_EVT	26/ext	PIN5	set	-	one-shot
TASK_t1	11/basic	PIN0	set	-	1.024 ms
TASK_t2	12/basic	PIN1	set	-	1.024 ms
TASK_t3	13/basic	PIN2	set	-	1.024 ms
TASK_t4	14/basic	PIN0	set	-	1.024 ms
TASK_t5	15/basic	PIN1	set	-	1.024 ms
TASK_t6	16/basic	PIN2	set	-	1.024 ms
TASK_t7	17/basic	PIN3	set	-	1.024 ms
TASK_h1	8/ext	PIN4	set	-	0.256 ms
TASK_h2	7/ext	-	-	-	0.512 ms
TASK_h3	6/ext	-	-	-	0.512 ms
TASK_h4	5/ext	PIN5	set	-	0.512 ms
TASK_h5	4/basic	-	-	-	1.024 ms
TASK_h6	3/basic	-	-	-	1.024 ms
TASK_h7	2/basic	-	-	-	1.024 ms
TASK_h8	1/basic	PIN6	set	-	1.024 ms
TASK_BGND	0/basic	PIN7	toggle		N/A

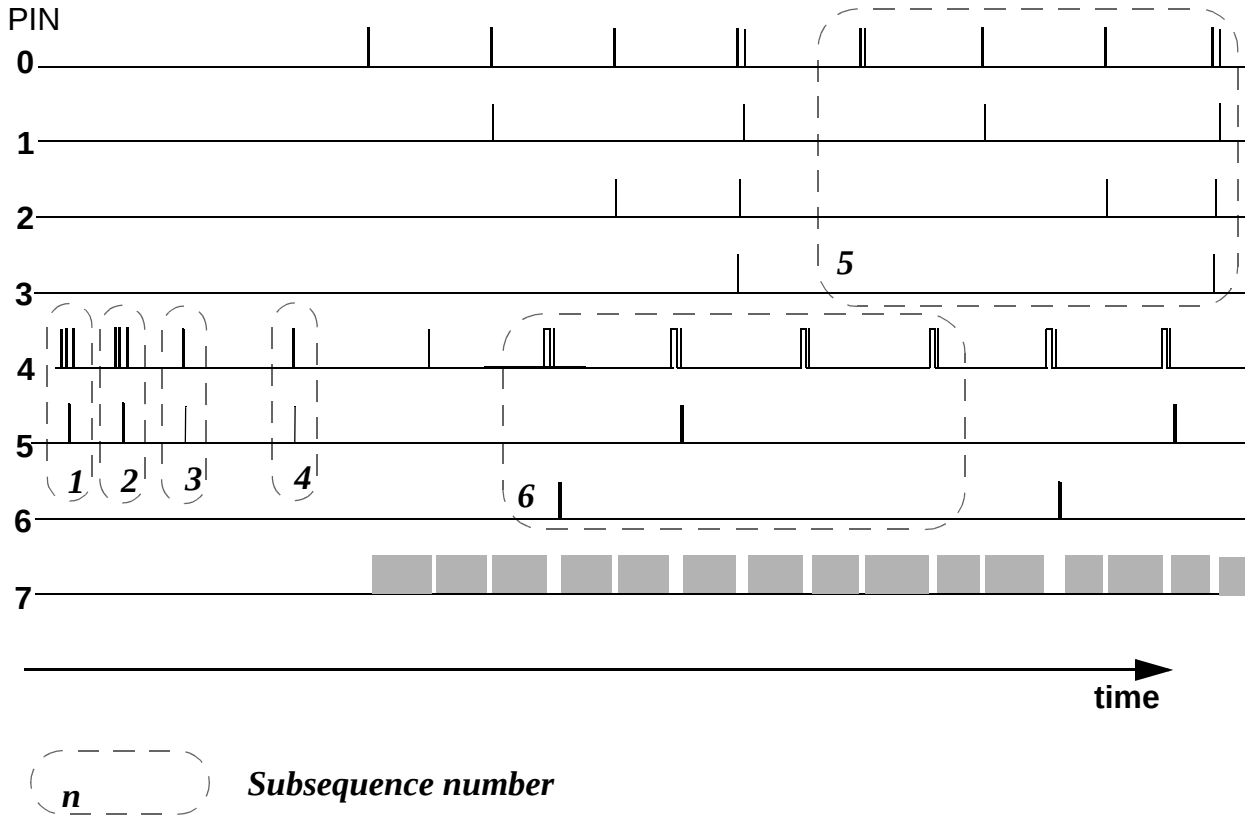
[a.] When any pin is set, all other are cleared (reset).

Execution Sequence

TASK_BGND is autostarted and performs initialization of measurement sequence.

Figure 2.6 shows the execution sequence for the application with pin states. The subsequences are marked with borders and the detailed descriptions of each measurement can be found below.

Figure 2.6 Execution Sequence for ECC1 Performance Benchmark



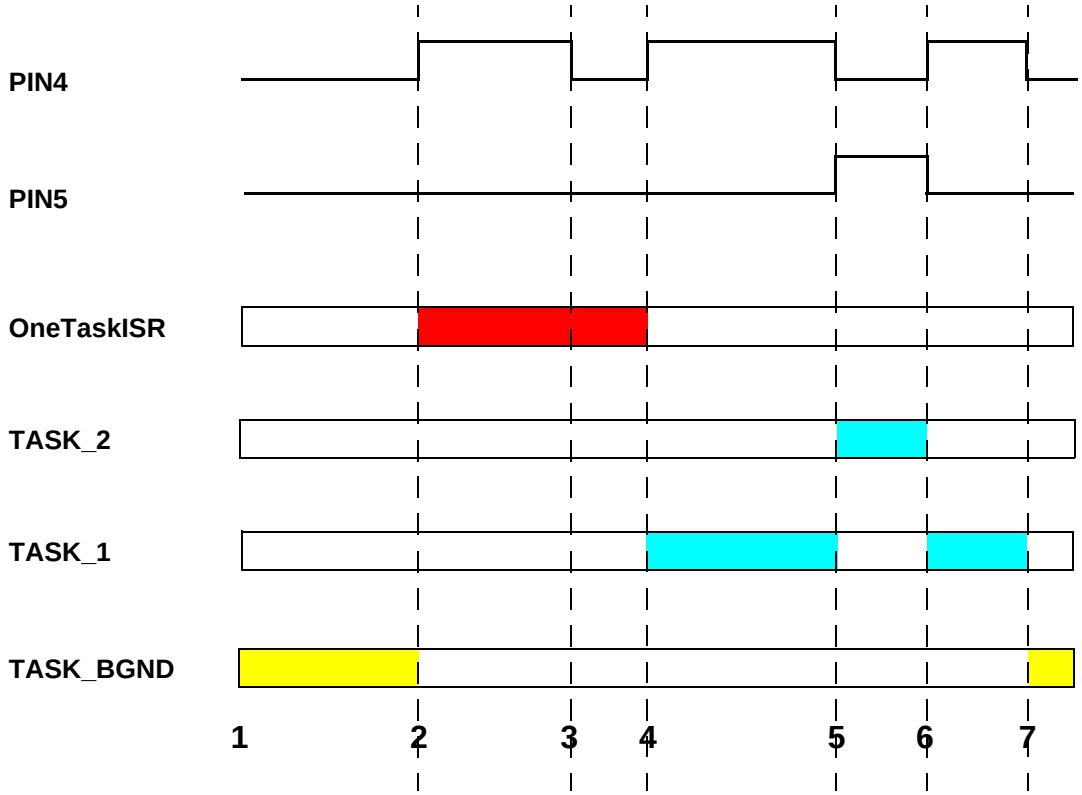
Subsequence to Measure Tasks Activate/Terminate Timings in ECC1 application

Measurements: ISR entry latency, switch time from ISR to tasks and between tasks

1. TASK_BGND initializes timer to raise an interrupt, processed by the OneTaskISR, in 0.5 ms, and then this task waits in loop until the flag variable has been set by the TASK_2.
2. OneTaskISR activates TASK_1.
3. OneTaskISR exits.
4. TASK_1 activates TASK_2.
5. TASK_2 starts and terminates.
6. After TASK_2 terminates, the control returns to TASK_1, and TASK_1 terminates.
7. After TASK_1 terminates, the control returns to TASK_BGND.

Figure 2.7 shows the execution subsequence for measuring tasks activate / terminate timings.

Figure 2.7 Task Activation/Termination Time Measurement



The tasks activate / terminate timings measurement results for CodeWarrior, WindRiver and GreenHills are indicated in the tables below.

Table 2.15 Task Activation/Termination Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Interrupt entry latency	- [a]	147	2,30	287	4,48	224	3,50	322	5,03
ActivateTask from ISR	2-3	92	1,44	88	1,38	137	2,14	136	2,13
From ISR to TASK_1	3-4	244	3,81	342	5,34	321	5,02	420	6,56
Activate to TASK_2	4-5	213	3,33	327	5,11	315	4,92	456	7,13
Terminate to TASK_1	5-6	177	2,77	294	4,59	255	3,98	351	5,48
Return to background	6-7	238	3,72	296	4,63	333	5,20	390	6,09

a. The interrupt entry latency is measured from time of interrupt request from hardware raised to the first command of user ISR.

Table 2.16 Task Activation/Termination Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Interrupt entry latency	- [a]	168	2,63	372	5,81	218	3,41	366	5,72
ActivateTask from ISR	2-3	109	1,70	108	1,69	164	2,56	163	2,55
From ISR to TASK_1	3-4	258	4,03	372	5,81	334	5,22	447	6,98
Activate to TASK_2	4-5	223	3,48	410	6,41	326	5,09	511	7,98
Terminate to TASK_1	5-6	179	2,80	288	4,50	244	3,81	351	5,48
Return to background	6-7	222	3,47	289	4,52	289	4,52	364	5,69

a. The interrupt entry latency is measured from time of interrupt request from hardware raised to the first command of user ISR.

Table 2.17 Task Activation/Termination Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Interrupt entry latency	- [a]	143	2,23	290	4,53	194	3,03	308	4,81
ActivateTask from ISR	2-3	95	1,48	97	1,52	136	2,13	137	2,14
From ISR to TASK_1	3-4	213	3,33	290	4,53	292	4,56	365	5,70
Activate to TASK_2	4-5	191	2,98	274	4,28	287	4,48	382	5,97
Terminate to TASK_1	5-6	152	2,38	224	3,50	205	3,20	274	4,28
Return to background	6-7	184	2,88	229	3,58	268	4,19	301	4,70

a. The interrupt entry latency is measured from time of interrupt request from hardware raised to the first command of user ISR.

.

Subsequence to Measure Tasks SetEvent/Terminate Timings

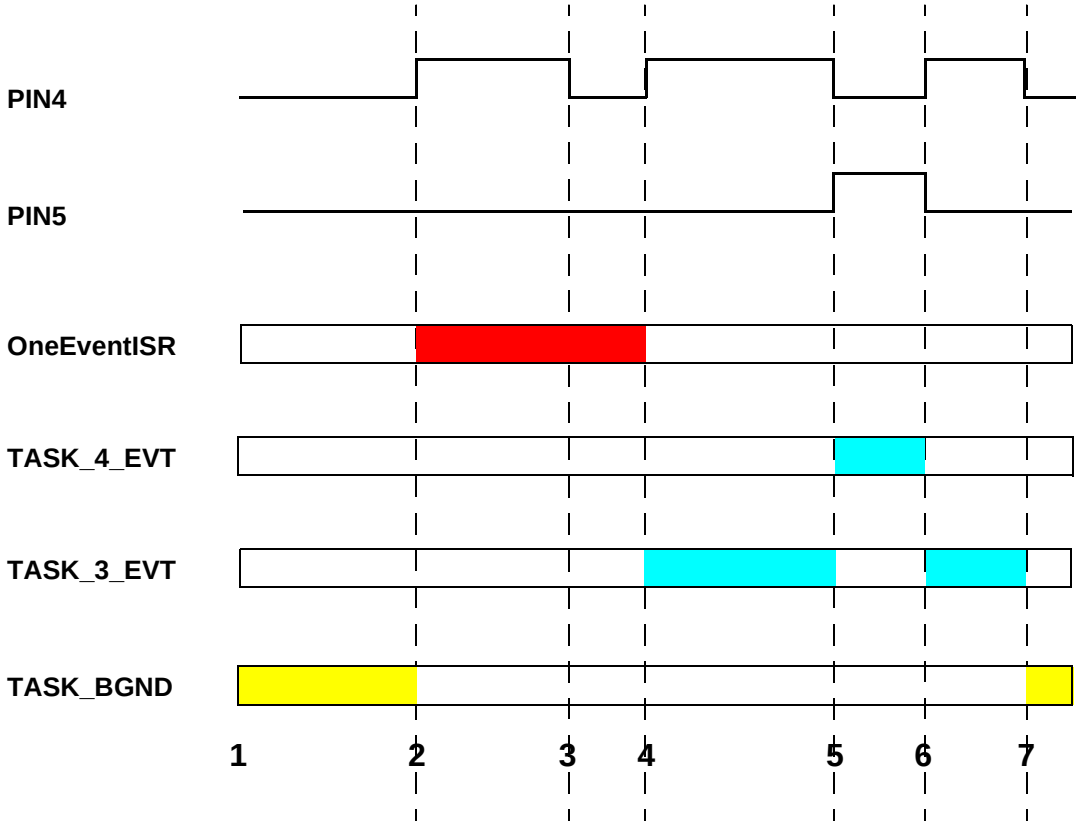
Measurements: ISR entry latency, switch time from ISR to tasks and between tasks by event setting

1. Tasks TASK_3_EVT and TASK_4_EVT are autostarted and switched to waiting state
2. TASK_BGND initializes timer to raise an interrupt, processed by the OneEventISR, in ms, and then this task waits in loop until the flag variable has been set by TASK_4_EVT
3. OneEventISR sets an event for TASK_3_EVT

4. OneEventISR exits
5. TASK_3_EVT sets an event TASK_4_EVT
6. TASK_4_EVT starts and terminates
7. After TASK_2 terminates, the control returns to TASK_3_EVT, and TASK_3_EVT terminates
8. After TASK_3_EVT terminates, the control returns to TASK_BGND

Figure 2.8 shows the execution subsequence for measuring task switch with event setting / termination timings.

Figure 2.8 Task Switch with Event Setting Time Measurement



The task switch with event setting / terminate timings measurement results for CodeWarrior, WindRiver and GreenHills are indicated in the tables below. . .

Table 2.18 Task Switch with Event Setting Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SetEvent within ISR	2-3	128	2,00	122	1,91	183	2,86	182	2,84
From ISR to TASK_3_EVT	3-4	284	4,44	394	6,16	360	5,63	476	7,44
SetEvent to TASK_4_EVT	4-5	292	4,56	434	6,78	398	6,22	556	8,69
Return to background	6-7	238	3,72	296	4,63	333	5,20	390	6,09

Table 2.19 Task Switch with Event Setting Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SetEvent within ISR	2-3	150	2,34	147	2,30	224	3,50	219	3,42
From ISR to TASK_3_EVT	3-4	311	4,86	437	6,83	366	5,72	494	7,72
SetEvent to TASK_4_EVT	4-5	326	5,09	512	8,00	417	6,52	613	9,58
Return to background	6-7	222	3,47	289	4,52	289	4,52	364	5,69

Table 2.20 Task Switch with Event Setting Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SetEvent within ISR	2-3	124	1,94	133	2,08	167	2,61	181	2,83
From ISR to TASK_3_EVT	3-4	244	3,81	342	5,34	313	4,89	402	6,28
SetEvent to TASK_4_EVT	4-5	270	4,22	374	5,84	351	5,48	467	7,30
Return to background	6-7	184	2,88	229	3,58	268	4,19	301	4,70

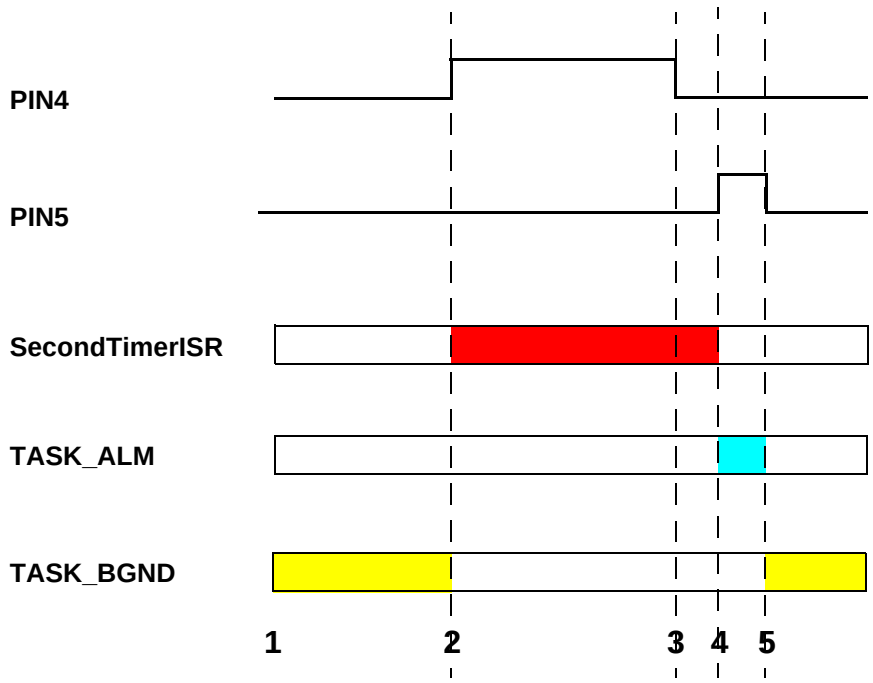
Subsequence to Measure Alarm Timings in ECC1 application

Measurements: Task activation by alarm expiration, switch time from ISR to task

1. TASK_BGND sets one short alarm ALARM_1 on Counter2 attached to the SecondTimerISR, and it waits in loop until the flag variable has been set by TASK_ALM
2. SecondTimerISR performs CounterTrigger service, an alarm expires and the task is activated by the alarm
3. SecondTimerISR exits
4. TASK_ALM starts and terminates
5. After TASK_ALM terminate, the control returns to TASK_BGND

Figure 2.9 shows the execution subsequence for measuring alarm timings.

Figure 2.9 Task Activation by Alarm Time Measurement in ECC1



The tasks activation by alarm expiration measurement results for CodeWarrior, WindRiver and GreenHills are indicated in the tables below. ..

Table 2.21 Task Activation by Alarm Time Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 alarm - Activate via Alarm	2-3	393	6,14	486	7,59	466	7,28	523	8,17
From ISR to TASK_ALM	3-4	236	3,69	331	5,17	310	4,84	412	6,44
Return to background	4-5	238	3,72	296	4,63	333	5,20	390	6,09

Table 2.22 Task Activation by Alarm Time Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 alarm - Activate via Alarm	2-3	376	5,88	530	8,28	415	6,48	515	8,05
From ISR to TASK_ALM	3-4	257	4,02	371	5,80	332	5,19	445	6,95
Return to background	4-5	222	3,47	289	4,52	289	4,52	364	5,69

Table 2.23 Task Activation by Alarm Time Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 alarm - Activate via Alarm	2-3	327	5,11	436	6,81	377	5,89	459	7,17
From ISR to TASK_ALM	3-4	208	3,25	285	4,45	286	4,47	363	5,67
Return to background	4-5	184	2,88	229	3,58	268	4,19	301	4,70

Subsequence to Measure Alarm with SetEvent Timings

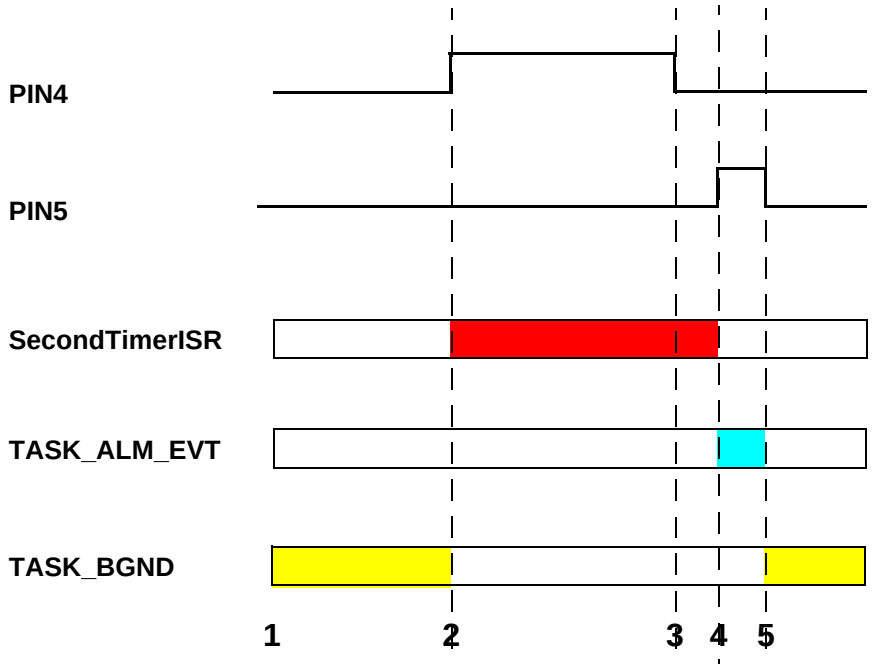
Measurements: Task event setting notification by alarm expiration, switch time from ISR to task

1. Task TASK_ALM_EVT is autostarted and switched to the waiting state
2. TASK_BGND sets one shot alarm ALARM_TWO on Counter2 attached to the SecondTimerISR, and it waits in loop until the flag variable has been set by TASK_ALM_EVT

3. SecondTimerISR performs CounterTrigger service, an alarm expires and the task is notified by the alarm with event setting
4. SecondTimerISR exits
5. TASK_ALM_EVT starts and terminates
6. After TASK_ALM_EVT terminates, the control returns to TASK_BGND

Figure 2.10 shows the execution subsequence for measuring alarm with event settings timings.

Figure 2.10 Task Notification by Alarm with Event Setting Time Measurement



The tasks notification by alarm with event settings measurement results ffor CodeWarrior, WindRiver and GreenHills are indicated in the tables below.

Table 2.24 Task Notification by Alarm with Event Setting timing (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: SetEvent via alarm	2-3	415	6,48	507	7,92	489	7,64	545	8,52
From ISR to TASK_ALM_EVT	3-4	280	4,38	390	6,09	356	5,56	472	7,38
Return to background	4-5	238	3,72	296	4,63	333	5,20	390	6,09

Table 2.25 Task Notification by Alarm with Event Setting timing (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: SetEvent via alarm	2-3	410	6,41	559	8,73	454	7,09	551	8,61
From ISR to TASK_ALM_EVT	3-4	309	4,83	435	6,80	364	5,69	492	7,69
Return to background	4-5	222	3,47	289	4,52	289	4,52	364	5,69

Table 2.26 Task Notification by Alarm with Event Setting timing (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: SetEvent via alarm	2-3	351	5,48	460	7,19	399	6,23	484	7,56
From ISR to TASK_ALM_EVT	3-4	244	3,81	343	5,36	316	4,94	402	6,28
Return to background	4-5	184	2,88	229	3,58	268	4,19	301	4,70

Subsequences to Measure Schedule Table and "Harmonic Alarms" Timings for ECC1 Application

This is the last subsequence and it is cycled. The Schedule Table and Harmonic Alarms measurements do not interfere because they do not happen at the same time.

TASK_BGND sets 8 alarms for tasks TASK_h[1...8] and initializes Schedule Table at such a moment that interrupts for the system and the second timer occur with 1 ms interval. These two measurements will be described separately.

Schedule Table for ECC1 Performance Measurement

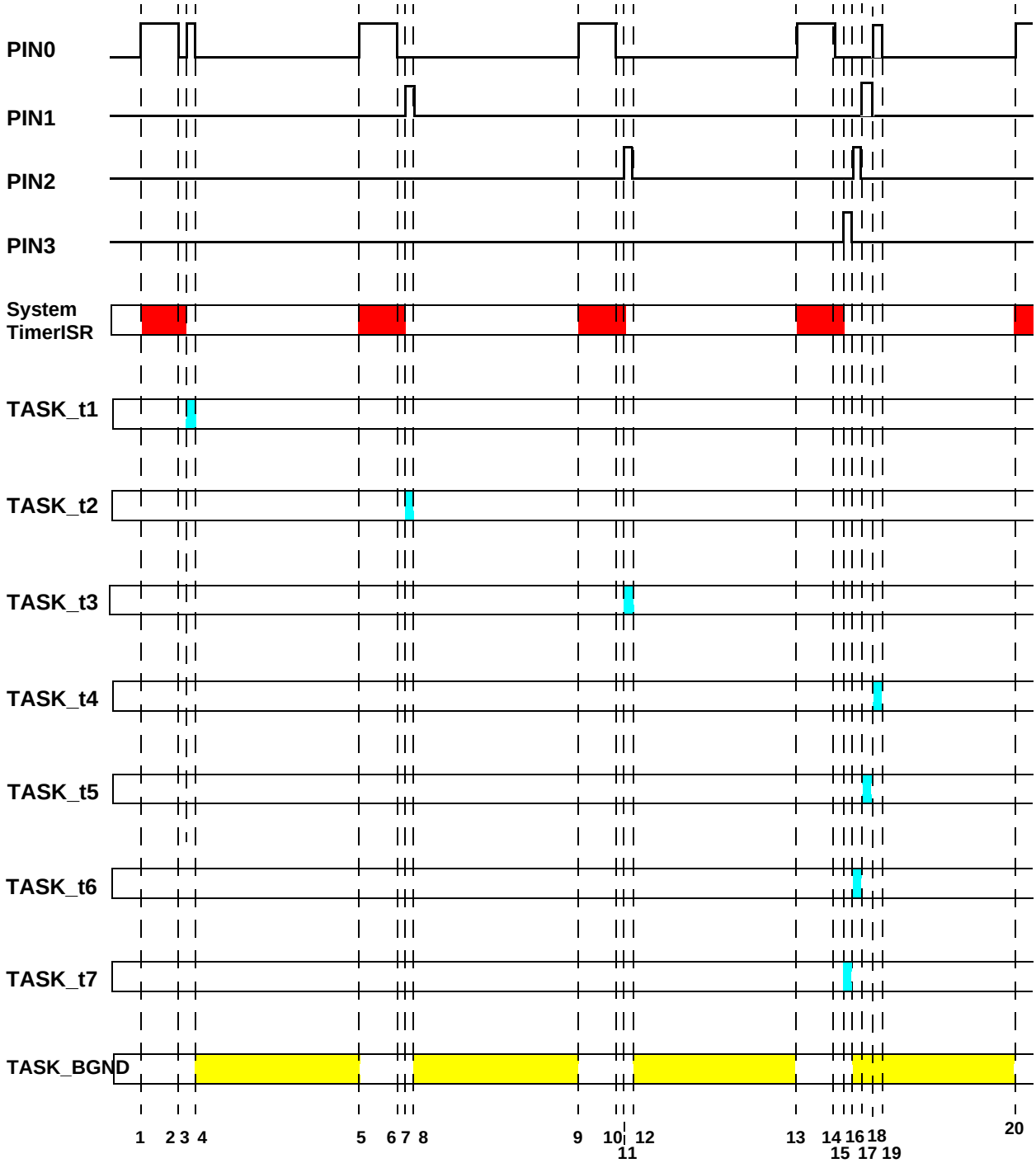
At points 1, 5, 9, 13, 20 the interrupt of SystemTimerISR occurs and the task corresponding to the appropriate step of the Schedule Table is activated. Point 1 corresponds to step 1, point 5 corresponds to step 2, point 9 corresponds to step 3, point 13 corresponds to steps 4, 5, 6, 7, and point 20 also corresponds to step 1. Every time the SystemTimerISR performs time scale processing before the next point.

The points 2, 6, 10, 14 correspond to the end of ISR code execution.

Every task activated by the Schedule Table calls only the TerminateTask and returns the control to TASK_BGND.

Figure 2.11 shows the execution subsequence for measuring the Schedule Table performance for the ECC1 application. The interval between adjacent system timer interrupts is ms, but the scale is unproportional because some of the measured intervals are insignificant compared to the time between interrupts.

Figure 2.11 Schedule Table for ECC1 Time Measurement



The Schedule Table performance measurement results for ECC1 application for CodeWarrior, WindRiver and GreenHills are indicated in the tables below. .

Table 2.27 Schedule Table Time for ECC1 Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Schedule Table ISR - 1 task activated	1-2, 5-6, 9-10	525	8,20	565	8,83	550	8,59	596	9,31
Schedule Table ISR - 4 tasks activated	13-14	761	11,89	801	12,52	797	12,45	843	13,17
From ISR to TASK_t1	2-3	223	3,48	304	4,75	285	4,45	376	5,88
From ISR to TASK_t2	6-7	223	3,48	304	4,75	285	4,45	376	5,88
From ISR to TASK_t3	10-11	223	3,48	304	4,75	285	4,45	376	5,88
From ISR to TASK_t7	14-15	223	3,48	304	4,75	285	4,45	376	5,88
TASK_t7 to TASK_t4[a]	15-18	364	5,69	706	11,03	651	10,17	939	14,67
Return to background	3-4, 7-8, 11-12, 18-19	238	3,72	296	4,63	333	5,20	390	6,09

[a.] Switch time between individual tasks are not measured

Table 2.28 Schedule Table Time for ECC1 Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Schedule Table ISR - 1 task activated	1-2, 5-6, 9-10	476	7,44	550	8,59	469	7,33	526	8,22
Schedule Table ISR - 4 tasks activated	13-14	742	11,59	830	12,97	727	11,36	793	12,39
From ISR to TASK_t1	2-3	221	3,45	313	4,89	289	4,52	385	6,02
From ISR to TASK_t2	6-7	223	3,48	311	4,86	287	4,48	383	5,98
From ISR to TASK_t3	10-11	221	3,45	313	4,89	289	4,52	385	6,02
From ISR to TASK_t7	14-15	223	3,48	314	4,91	290	4,53	386	6,03
TASK_t7 to TASK_t4[a]	15-18	347	5,42	724	11,31	634	9,91	972	15,19
Return to background	3-4, 7-8, 11-12, 18-19	222	3,47	289	4,52	289	4,52	364	5,69

[a.] Switch time between individual tasks are not measured

Table 2.29 Schedule Table Time for ECC1 Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
Schedule Table ISR - 1 task activated	1-2, 5-6, 9-10	446	6,97	506	7,91	450	7,03	503	7,86
Schedule Table ISR - 4 tasks activated	13-14	664	10,38	745	11,64	674	10,53	730	11,41
From ISR to TASK_t1	2-3	192	3,00	262	4,09	249	3,89	322	5,03
From ISR to TASK_t2	6-7	192	3,00	262	4,09	249	3,89	322	5,03
From ISR to TASK_t3	10-11	192	3,00	262	4,09	249	3,89	322	5,03
From ISR to TASK_t7	14-15	192	3,00	262	4,09	249	3,89	322	5,03
TASK_t7 to TASK_t4[a]	15-18	303	4,73	539	8,42	526	8,22	725	11,33
Return to background	3-4, 7-8, 11-12, 18-19	184	2,88	229	3,58	268	4,19	301	4,70

[a.] Switch time between individual tasks are not measured

"Harmonic Alarms" for ECC1 Performance Measurement

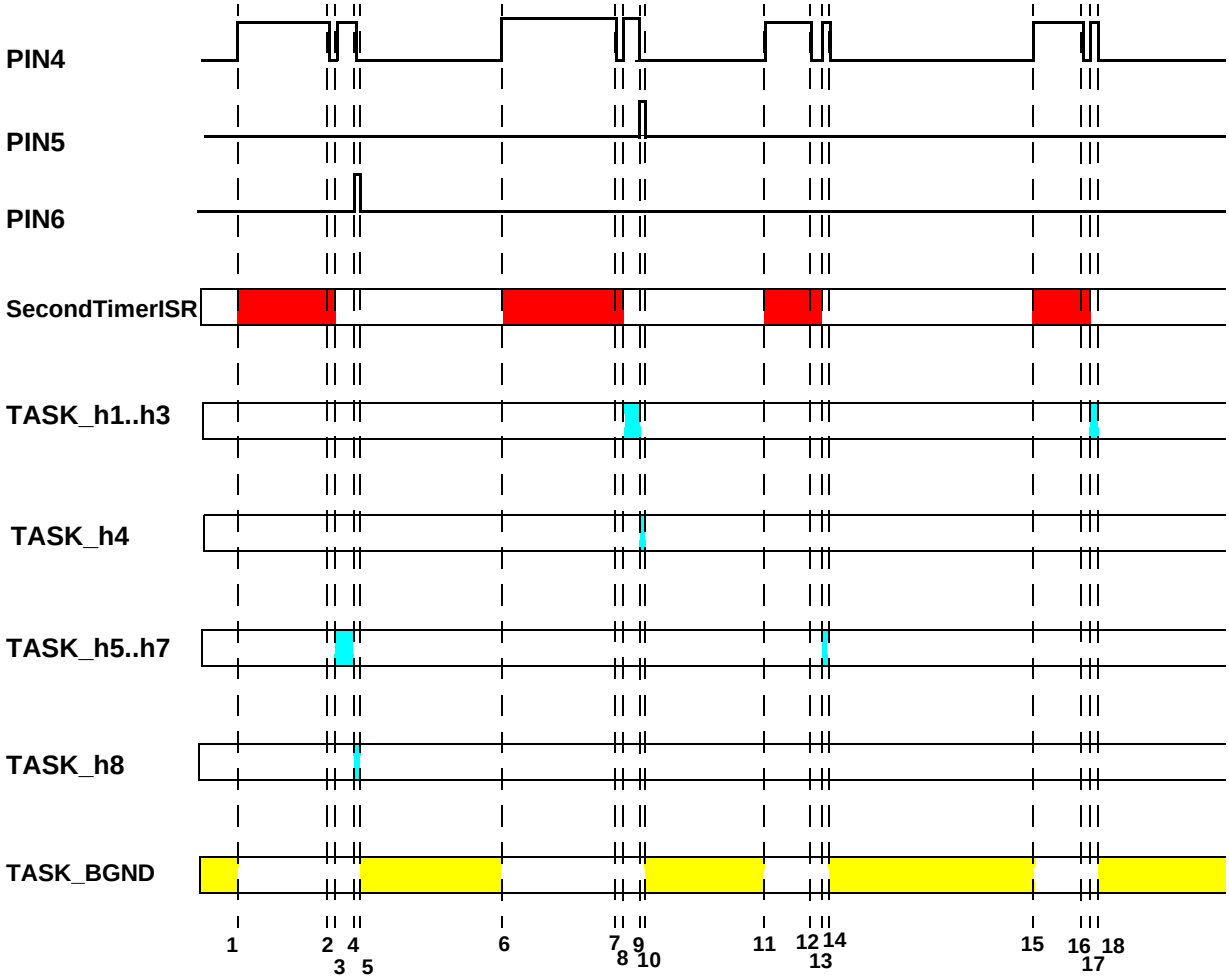
At points 1, 6, 11, 15 the interrupt of the SecondTimerISR occurs and the alarms attached to the Counter2 are processed. At point 1 four alarms expire and TASK_h5..h8 are activated. At point 6 four alarms expire and TASK_h1..h4 are notified by events setting. At point 11 one alarm expires and TASK_h5 is activated. At point 15 one alarm expires and TASK_h1 is notified by event setting. Every time the SecondTimerISR performs the counter and alarms processing before the next point.

Points 2, 7, 12, 16 correspond to the end of ISR code execution.

Every task activated by alarms only calls the TerminateTask and returns the control to TASK_BGND.

Figure 2.12 shows the execution subsequence for measuring "Harmonic alarms" performance for the ECC1 application. The interval between adjacent system timer interrupts is ms, but the scale is unproportional because some of the measured intervals are insignificant compared to the time between interrupts.

Figure 2.12 "Harmonic Alarms" for ECC1 Time Measurement



The "Harmonic alarms" performance measurement results for ECC1 application for CodeWarrior, WindRiver and GreenHills are indicated in the tables below.

Table 2.30 "Harmonic alarms" Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 task activated via alarm[a]	11-12	487	7,61	573	8,95	562	8,78	620	9,69
SecondTimerISR: 1 event set via alarm	15-16	504	7,88	589	9,20	577	9,02	636	9,94

Table 2.30 "Harmonic alarms" Measurement Results (CW)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 4 tasks activated via alarms	1-2	672	10,50	761	11,89	748	11,69	806	12,59
SecondTimerISR: 4 events set via alarms	6-7	739	11,55	826	12,91	820	12,81	879	13,73
From ISR to TASK_h1	7-8, 16-17	278	4,34	387	6,05	352	5,50	470	7,34
TASK_h1 to TASK_h4	8-9	897	14,02	1338	20,91	1210	18,91	1645	25,70
From ISR to TASK_h5	2-3, 12-13	236	3,69	331	5,17	310	4,84	412	6,44
TASK_h5 to TASK_h8	3-4	364	5,69	708	11,06	651	10,17	937	14,64

a. The time for the counter processing depends on the number of the alarms actually set. Compare timing for the SecondTimerISR in when only one alarm is set.

Table 2.31 "Harmonic alarms" Measurement Results (WindRiver)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 task activated via alarm[a]	11-12	500	7,81	655	10,23	524	8,19	643	10,05
SecondTimerISR: 1 event set via alarm	15-16	528	8,25	680	10,63	556	8,69	672	10,50
SecondTimerISR: 4 tasks activated via alarms	1-2	756	11,81	922	14,41	771	12,05	889	13,89
SecondTimerISR: 4 events set via alarms	6-7	880	13,75	1030	16,09	905	14,14	1016	15,88
From ISR to TASK_h1	7-8, 16-17	310	4,84	435	6,80	365	5,70	493	7,70
TASK_h1 to TASK_h4	8-9	974	15,22	1460	22,81	1332	20,81	1861	29,08
From ISR to TASK_h5	2-3, 12-13	256	4,00	370	5,78	331	5,17	446	6,97
TASK_h5 to TASK_h8	3-4	348	5,44	728	11,38	638	9,97	974	15,22

a. The time for the counter processing depends on the number of the alarms actually set. Compare timing for the SecondTimerISR in Table 2.22 when only one alarm is set.

Table 2.32 "Harmonic alarms" Measurement Results (GHS)

Measurement	Time points	SC1		SC2		SC3		SC4	
		Cycles	µs	Cycles	µs	Cycles	µs	Cycles	µs
SecondTimerISR: 1 task activated via alarm[a]	11-12	434	6,78	557	8,70	516	8,06	569	8,89
SecondTimerISR: 1 event set via alarm	15-16	450	7,03	574	8,97	535	8,36	582	9,09
SecondTimerISR: 4 tasks activated via alarms	1-2	619	9,67	779	12,17	722	11,28	788	12,31
SecondTimerISR: 4 events set via alarms	6-7	707	11,05	869	13,58	807	12,61	878	13,72
From ISR to TASK_h1	7-8, 16-17	242	3,78	339	5,30	317	4,95	400	6,25
TASK_h1 to TASK_h4	8-9	804	12,56	1075	16,80	1021	15,95	1328	20,75
From ISR to TASK_h5	2-3, 12-13	207	3,23	288	4,50	284	4,44	362	5,66
TASK_h5 to TASK_h8	3-4	301	4,70	538	8,41	528	8,25	726	11,34

a. The time for the counter processing depends on the number of the alarms actually set. Compare timing for the SecondTimerISR in when only one alarm is set.

3

Memory and Overhead Benchmarks

This chapter of the manual describes some typical applications for measuring memory consumption and system overhead. Configuration, actions taken, and measurements are described in detail for each benchmark application.

This chapter includes the following sections:

System Overhead and Memory Consumption Measurements

The following method is used for measuring a system overhead:

The background task is autostarted and disables interrupts from the system timer. Then the background task is synchronized with the value of the Counter2 which is incremented on each timer interrupt configured as the second system timer. The timer is configured to tick every ms.

The background task increments a variable for the time about 500 ms with System Timer interrupts disabled. After that the timer interrupts are enabled and the system timer starts working, the alarms expire and tasks are activated. The background task increments another variable in the background. The system overhead is calculated by comparing the value of the variable counted when the system timer interrupt was disabled and the

value of the variable counted when system timer worked and tasks were activated.

Information from the map file is used to calculate memory consumption. ROM usage is calculated as a sum of the size of code and constant“.vects”, “.ostext”, “.text”, “.osrodata”, “.rodata”. ROM usage for OS code, Application code, OS constants and vector table are indicated separately. RAM usage is calculated as a sum of the size of “.sdata”, “.sbss”and “.bss” sections, excluding memory allocated for stacks. But the size of OS internal stacks for exception handlers is included into RAM usage number. Stack usage is measured inside application after overhead measurement finished.

Body Benchmark Application for BCC1

The benchmark application is based on a typical cycle (period) of body electronics activities. There are ten tasks, every 20 ms one of them is activated via an alarm, connected to the system counter. The system timer ticks every millisecond, and after every 20th tick a task is activated. Every millisecond the timer tick activates ISR, and every 20 ms one task is activated.

The files of this benchmark are located in a `MEMORY\BODY\BCC1` directory.

To execute the benchmark application in a Lauterbach debugger take the following actions (change <n> to 1..4 according to SC):

- Open the file bencmark\memory\common\test.cmm and change the line with <&suff="_gh"> - set it “_cw” for CodeWarrior or “_db” for WindRiver if required.
- Start the debugger
- Open `MEMORY\ECC1\SCn\pecc1sc<n>.cmm` debugger configuration file via menu “File -> Run Batch file”
- Wait until the benchmark executable is loaded, then starts execution and then stops.
- read results in the log window.

Configuration

The following OS configuration is used for the benchmark application:

- STANDARD OS status
- DEBUG_LEVEL is equal to 0
- No hooks are defined
- Full preemptive scheduler
- ResourceScheduler support is off
- FastTerminate is set to TRUE
- FastScheduler is set to TRUE

Clock frequency is set to 64 MHz

The benchmark application consists of the following OSEK OS objects:

- SysTimer - a system timer driven by the timer interrupt, which is configured to generate interrupt approximately every millisecond. A set of alarms is attached to the system timer counter and timer is started after timer interrupts are enabled by the TASK_BGND task
- TASK1...TASK10 - tasks activated by the alarms attached to the system timer Counter1. Each task starts and calls only the TerminateTask, thus returning the control to TASK_BGND
- TASK_BGND - a background task with the lowest priority, performs initialization of the measurements sequences and calculation of the system overhead
- ALARM1...ALARM10 - alarms that activate TASK1...10

Body BCC1 Benchmark Measurements Results

The measured system overhead for the BCC1 Body benchmark is shown in the table below:

Scalability Class	SC1	SC2	SC3	SC4
OS overhead for CodeWarrior	0,66%	0,85%	0,80%	0,89%
OS overhead for GreenHills	0,73%	0,88%	0,80%	0,90%
OS overhead for WindRiver	0,69%	0,94%	0,78%	0,95%

The memory consumption is presented in the tables below. .

Table 3.1 BCC1 Benchmark Memory Consumption for CodeWarrior

Memory	Size (bytes)			
	SC1	SC2	SC3	SC4
OS code	2244	3832	9176	10488
Application code	672	2668	2764	2760
Constants	1080	1224	1668	1756
Vector table	292	292	292	292
Compiler libraries and init code	784	796	796	796
Total ROM	5072	8812	14696	16092
Main Stack usage	360	184	168	184
Tasks (TASKn / TASKBGND) Stack usage	N/A	104/264	88/248	104/264
OS and Application variables RAM	300	1800	2368	2616

Table 3.2 BCC1 Benchmark Memory Consumption for GreenHills

Memory	Size (bytes)			
	SC1	SC2	SC3	SC4
OS code	2166	3392	8794	9812
Application code	712	1536	1624	1600
Constants	1080	1224	1660	1752
Vector table	292	292	292	292
Compiler libraries and init code	2232	2232	2232	2232
Total ROM	6482	8676	14602	15688
Main Stack usage	368	256	256	256
Tasks (TASKn / TASKBGND) Stack usage	N/A	72/200	72/200	72/200
OS and Application variables RAM	308	1812	2360	2608

Table 3.3 BCC1 Benchmark Memory Consumption for WindRiver

Memory	Size (bytes)			
	SC1	SC2	SC3	SC4
OS code	3664	5888	8764	9928
Application code	692	1500	1588	1576
Constants	1080	1224	1660	1752

Table 3.3 BCC1 Benchmark Memory Consumption for WindRiver

Memory	Size (bytes)			
	SC1	SC2	SC3	SC4
Vector table	292	292	292	292
Compiler libraries and init code	760	760	760	760
Total ROM	6488	9664	13064	14308
Main Stack usage	368	208	192	224
Tasks (TASKn / TASKBGND) Stack usage	N/A	136/264	152/264	184/296
OS and Application variables RAM	310	1814	2368	2612

Body Benchmark Application for ECC1

The benchmark application is based on a typical cycle (period) of body electronics activities. There are ten tasks, every 20 ms an alarm, connected to the system counter, sets an event for one of them and a task switched from waiting state.

The files of this benchmark are located in `MEMORY\BODY\ECC1` directory.

To execute the benchmark application in a Lauterbach debugger take the following actions (change <n> to 1..4 according to SC):

- Open the file bencmark\memory\common\test.cmm and change the line with <&suff="_gh"> - set it “_cw” for CodeWarrior or “_db” for WindRiver if required.
- Start the debugger
- Open `MEMORY\ECC1\SCn\pecc1sc<n>.cmm` debugger configuration file via menu “File -> Run Batch file”
- Wait until the benchmark executable is loaded, then starts execution and then stops.
- read results in the log window.

Configuration

The following OS configuration is used for the benchmark application:

- STANDARD OS status

- Conformance class is ECC1
- DEBUG_LEVEL is equal to 0
- No hooks are defined
- Full preemptive scheduler
- ResourceScheduler support disabled
- FastTerminate is set to TRUE
- FastScheduler is set to TRUE

Clock frequency is 64 MHz

The benchmark application consists of the following OSEK OS objects:

- SysTimer - a system timer driven by the timer interrupt, which is configured to generate interrupt approximately every millisecond. A set of alarms is attached to the system timer counter and timer is started after timer interrupts are enabled by the TASK_BGND task
- TASK1...TASK10 - extended tasks are autostarted and switched to waiting state, they get running after an alarm attached to the system timer Counter1 sets an event for the task. Each task performs an endless loop clearing its own event and then waits for it
- TASK_BGND - a background task with the lowest priority, performs initialization of the measurements sequences and calculation of the system overhead
- ALARM1...ALARM10 - alarms that set events for TASK1...10
- EVT1...EVT10 - events that are set by alarms

Body ECC1 Benchmark Measurements Results

The measured system overhead for the ECC1 Body benchmark is shown in the table below: :

Scalability Class	SC1	SC2	SC3	SC4
OS overhead for CodeWarrior	0,70%	0,83%	0,78%	0,88%
OS overhead for GreenHills	0,74%	0,85%	0,82%	0,95%
OS overhead for WindRiver	0,72%	0,91%	0,76%	0,92%

The memory consumption is presented in the tables below

Table 3.4 ECC1 Benchmark Memory Consumption for CodeWarrior

Memory	Size (bytes)			
	SC1	SC2	SC3	SC4
OS code	2872	4064	9996	11584
Application code	2692	2824	2956	2948
Constants	1160	1224	1684	1772
Vector table	292	292	292	292
Compiler library and init code	720	796	796	796
Total ROM	7736	9200	15724	17392
Main Stack usage	344	184	168	184
Tasks (TASKn / TASKBGND) Stack usage	72	136/280	120/264	136/280
OS and Application variables RAM	1632	1896	2464	2712

Table 3.5 ECC1 Benchmark Memory Consumption for GreenHills

Memory	Size (bytes)			
	SC1	SC2	SC3	SC4
OS code	2740	3684	9642	11098
Application code	1578	1614	1702	1692
Constants	1156	1224	1676	1768
Vector table	292	292	292	292
Compiler library and init code	2232	2232	2232	2232
Total ROM	7998	9046	15544	17082
Main Stack usage	352	256	256	256
Tasks (TASKn / TASKBGND) Stack usage	56	104/200	104/200	104/200
OS and Application variables RAM	1636	1908	2456	2704

Table 3.6 ECC1 Benchmark Memory Consumption for WindRiver

Memory	Size (bytes)			
	SC1	SC2	SC3	SC4
OS code	4916	6644	9556	11032
Application code	1596	1632	1728	1712

Table 3.6 ECC1 Benchmark Memory Consumption for WindRiver

Memory	Size (bytes)			
	SC1	SC2	SC3	SC4
Constants	1156	1224	1676	1768
Vector table	292	292	292	292
Compiler library and init code	760	760	760	760
Total ROM	8720	10552	14012	15564
Main Stack usage	352	208	192	224
Tasks (TASKn / TASKBGND) Stack usage	72	152/280	152/280	184/312
OS and Application variables RAM	1638	1910	2460	2708