Application Note

Integrating SEGGER
SystemView into an OS

AN08003
1.00
0
Date: January 18, 2018

A product of SEGGER Microcontroller GmbH

www.segger.com

Contact address

SEGGER Microcontroller GmbH

In den Weiden 11
D-40721 Hilden

Germany

Tel.	+49 2103-2878-0
Fax.	+49 2103-2878-28
E-mail:	`support@segger.com`
Internet:	`www.segger.com`

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: January 18, 2018

Revision	Date	By	Description
0	151109	JL	Initial Version.

Table of contents

Chapter 1

Integrating SEGGER SystemView into an OS

This application note describes how to integrate SEGGER SystemView into an (RT)OS to enable monitoring OS and task execution.

SEGGER SystemView is a real-time recording and visualization tool to gain a deep understanding of the runtime behavior of an application, going far beyond what debuggers are offering. This is particularly advantageous when developing and working in complex systems with multiple tasks and interrupts, to analyze problems and to find inefficiencies.

SEGGER SystemView can be used to to monitor the execution of an embedded system with an OS. The following sections describe how to integrate SystemView into an OS by instrumenting it with the SYSTEMVIEW module.

Chapter 2

Instrumenting the OS core

The mandatory part of integrating SystemView into an OS is to instrument the OS core, that is the OS Scheduler and routines which handle task activity and execution.

In order to be able to visualize task execution and context switches with SystemViewer, the SYSTEMVIEW module must record these events, which is done by adding SYSTEMVIEW event recording functions at appropriate code locations in the OS core.

Some OSes already include functionality to instrument the OS for tracing which can be used by SystemView. This could for example be possible by passing a trace API to the OS or by defining trace function macros.

If the OS does not already include this functionality, SYSTEMVIEW can either be included directly in the OS or by adding the methods mentioned above.

The examples in the following sections are pseudo-code to illustrate when to call SYSTEMVIEW functions. The methods mentioned above could be used in these places, too.

2.1 SYSTEMVIEW OS Callback API

Upon initialization a pointer to a `SEGGER_SYSVIEW_OS_API` struct with callbacks which are called by SYSTEMVIEW on specific events to retrieve information from the OS is passed to SYSTEMVIEW.

The OS API and its members are optional, but should be passed to get all information in SystemView.

2.1.1 pfGetTime

Description

Get the system time, i.e. the time since starting the system in microseconds.

If `pfGetTime` is `NULL` SystemViewer can show timestamps relative to the start of recording only.

Prototype

```
U64 (*pfGetTime) (void);
```

2.1.2 pfSendTaskList

Description

Record the entire task list via `SEGGER_SYSVIEW_SendTaskInfo()`.

If `pfSendTaskList` is `NULL` SystemViewer might only get task information of tasks which are newly created while recording. `pfSendTaskList` is called periodically to keep track on the current task list.

Prototype

```
void (*pfSendTaskList) (void);
```

Example

```
void cbSendTaskList(void) {
  SEGGER_SYSVIEW_TASKINFO Info;
  OS_TASK*                pTask;

  OS_EnterRegion(); // Disable scheduling to make sure the task list does not change.
  for (pTask = OS_Global.pTask; pTask; pTask = pTask->pNext) {
    //
    // Fill all elements with 0 to allow extending the structure
    // in future version without breaking the code.
    //
    memset(&Info, 0, sizeof(Info));
    //
    // Fill elements with current task information
    //
    Info.TaskID     = (U32)pTask;
    Info.sName      = pTask->Name;
    Info.Prio       = pTask->Priority;
    Info.StackBase  = (U32)pTask->pStackBot;
    Info.StackSize  = pTask->StackSize;
    //
    // Record current task information
    //
    SEGGER_SYSVIEW_SendTaskInfo(&Info);
  }
  OS_LeaveRegion(); // Enable scheduling again.
}
```

2.2 Recording task activity

SystemView can record a set of pre-defined system events, which provide the main information of the system and OS activity. The SYSTEMVIEW API provides functions for these events which should be added to the OS when an event happens.

The usual events are:

Event	Description
Task Create	A new task is created.
Task Start Ready	A task is marked as ready to start or resume execution.
Task Start Exec	A task is activated, it starts or resumes execution.
Task Stop Ready	A task is blocked or suspended.
Task Stop Exec	A task stops execution.
System Idle	No task is executing, the system goes into Idle state.

2.2.1 Task Create

A new task is created.

Task Create events happen when a task is created by the system.

On Task Create events call `SEGGER_SYSVIEW_OnTaskCreate()` with the Id of the new task. Additionally it is recommended to record the task information of the new task with `SEGGER_SYSVIEW_SendTaskInfo()`.

Example

```
void OS_CreateTask(TaskFunc* pF, unsigned Prio, const char* sName, void* pStack) {
  SEGGER_SYSVIEW_TASKINFO Info;
  OS_TASK*                pTask;  // Pseudo struct to be replaced

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskCreate((unsigned)pTask);
  memset(&Info, 0, sizeof(Info));
  //
  // Fill elements with current task information
  //
  Info.TaskID     = (U32)pTask;
  Info.sName      = pTask->Name;
  Info.Prio       = pTask->Priority;
  Info.StackBase  = (U32)pTask->pStack;
  Info.StackSize  = pTask->StackSize;
  SEGGER_SYSVIEW_SendTaskInfo(&Info);
}
```

2.2.2 Task Start Ready

A task is marked as ready to start or resume execution.

Task Start Ready events can for example happen, when the delay time of the task expired, or when a resource the task was waiting for is available, or when an event was triggered.

On Task Start Ready events call `SEGGER_SYSVIEW_OnTaskStartReady()` with the Id of the task which has become ready.

Example

```
int OS_HandleTick(void) {
  int TaskReady = 0;   // Pseudo variable indicating a task is ready
  [OS specific code ...]

  if (TaskReady) {
    SEGGER_SYSVIEW_OnTaskStartReady((unsigned)pTask);
  }
}
```

2.2.3 Task Start Exec

A task is activated, it starts or resumes execution.

Task Start Exec events happen when the context is about to be switched to the activated task. This is normally done by the Scheduler when there is a ready task.

On Task Start Exec events call `SEGGER_SYSVIEW_OnTaskStartExec()` with the Id of the task which will execute.

Example

```
void OS_Switch(void) {

  [OS specific code ...]

  //
  // If a task is activated
  //
  SEGGER_SYSVIEW_OnTaskStartExec((unsigned)pTask);
  //
  // Else no task activated, go into idle state
  //
  SEGGER_SYSVIEW_OnIdle()
}
```

2.2.4 Task Stop Ready

A task is blocked or suspended.

Task Stop Ready events happen when a task is suspended or blocked, for example because it delays for a specific time, or when it tries to claim a resource which is in use by another task, or when it waits for an event to happen. When a task is suspended or blocked the Scheduler will activate another task or go into idle state.

On Task Stop Ready events call `SEGGER_SYSVIEW_OnTaskStopReady()` with the Id of the task which is blocked and a ReasonId which can indicate why the task is blocked.

Example

```
void OS_Delay(unsigned NumTicks) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopReady(OS_Global.pCurrentTask, OS_CAUSE_WAITING);
}
```

2.2.5 Task Stop Exec

A task stops execution.

Task Stop Exec events happen when a task finally stops execution, for example when it has done its job and terminates.

On Task Stop Ready events call `SEGGER_SYSVIEW_OnTaskStopExec()` to record the current task as stopped.

Example

```
void OS_TerminateTask(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopExec();
}
```

2.2.6 System Idle

No task is executing, the system goes into Idle state.

System Idle events happen, when a task is suspended or stopped and no other task is ready. The system can switch into an idle state to save power, wait for an interrupt or a task to become ready.

In some OSes Idle is handled by an additional task. In this case it is recommended to record System Idle events, when the Idle task is activated, too.

Time spent in Idle state is displayed as not CPU Load in SystemViewer.

On System Idle events call `SEGGER_SYSVIEW_OnIdle()`.

Example

```
void OS_Switch(void) {

  [OS specific code ...]

  //
  // If a task is activated
  //
  SEGGER_SYSVIEW_OnTaskStartExec((unsigned)pTask);
  //
  // Else no task activated, go into idle state
  //
  SEGGER_SYSVIEW_OnIdle()
}
```

2.3 Interrupts

SystemView can record enter interrupt and exit interrupt events. The SYSTEMVIEW API provides functions for these events which should be added to the OS when it provides functions to mark interrupt execution.

When the Scheduler is controlled by interrupts, i.e. the SysTick interrupt, the exit interrupt event should distinguish between resuming normal execution or switching into the Scheduler and call the appropriate SYSVIEW function.

2.3.1 Enter Interrupt

When the OS provides a function to inform the OS that interrupt code is executing, to be called at the start of an Interrupt Service Routine (ISR), the OS function should call `SEGGER_SYSVIEW_RecordEnterISR()` to record the Enter Interrupt event.

When the OS does not provide an enter interrupt function, or the ISR does not call it, it is the user's responsibility to call `SEGGER_SYSVIEW_RecordEnterISR()` to be able to record interrupt execution.

`SEGGER_SYSVIEW_RecordEnterISR()` automatically retrieves the interrupt ID via the `SEGGER_SYSVIEW_GET_INTERRUPT_ID()` function macro as defined in `SEGGER_SYSVIEW_Conf.h`.

Example

```
void OS_EnterInterrupt(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_RecordEnterISR();
}
```

2.3.2 Exit Interrupt

When the OS provides a function to inform the OS that interrupt code has executed, to be called at the and of an Interrupt Service Routine (ISR), the OS function should call:

- `SEGGER_SYSVIEW_RecordExitISR()` when the system will resume normal execution.
- `SEGGER_SYSVIEW_RecordExitISRToScheduler()` when the interrupt caused a context switch.

Example

```
void OS_ExitInterrupt(void) {

  [OS specific code ...]
  //
  // If the interrupt will switch to the Scheduler
  //
  SEGGER_SYSVIEW_RecordExitISRToScheduler();
  //
  // Otherwise
  //
  SEGGER_SYSVIEW_RecordExitISR();
}
```

2.3.3 Example ISRs

The following two examples show how to record interrupt execution with SystemView with OS interrupt handling and without.

Example with OS handling

```
void Timer_Handler(void) {
  //
  // Inform OS about start of interrupt execution
  // (records SystemView Enter Interrupt event).
  //
  OS_EnterInterrupt();
  //
  // Interrupt functionality could be here
  //
  APP_TimerCnt++;
  //
  // Inform OS about end of interrupt execution
  // (records SystemView Exit Interrupt event).
  //
  OS_ExitInterrupt();
}
```

Example without OS handling

```
void ADC_Handler(void) {
  //
  // Explicitly record SystemView Enter Interrupt event.
  // Should not be called in high-frequency interrupts.
  //
  SEGGER_SYSVIEW_RecordEnterISR();
  //
  // Interrupt functionality could be here
  //
  APP_ADCValue = ADC.Value;
  //
  // Explicitly record SystemView Exit Interrupt event.
  // Should not be called in high-frequency interrupts.
  //
  SEGGER_SYSVIEW_RecordExitISR();
}
```

2.4 Sample pseudo-code OS integration

```
/*********************************************************************
*            (c) 1995 - 2018 SEGGER Microcontroller GmbH             *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************
-------------------------- END-OF-HEADER -----------------------------

Purpose : Pseudo-code OS with SEGGER SystemView integration.
*/

/*********************************************************************
*
*       OS_CreateTask()
*
*  Function description
*    Create a new task and add it to the system.
*/
void OS_CreateTask(TaskFunc* pF, unsigned Prio, const char* sName, void* pStack) {
  SEGGER_SYSVIEW_TASKINFO Info;
  OS_TASK*                pTask;  // Pseudo struct to be replaced

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskCreate((unsigned)pTask);
  memset(&Info, 0, sizeof(Info));
  //
  // Fill elements with current task information
  //
  Info.TaskID     = (U32)pTask;
  Info.sName      = pTask->Name;
  Info.Prio       = pTask->Priority;
  Info.StackBase  = (U32)pTask->pStack;
  Info.StackSize  = pTask->StackSize;
  SEGGER_SYSVIEW_SendTaskInfo(&Info);
}

/*********************************************************************
*
*       OS_TerminateTask()
*
*  Function description
*    Terminate a task and remove it from the system.
*/
void OS_TerminateTask(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopExec();
}

/*********************************************************************
*
*       OS_Delay()
*
*  Function description
*    Delay and suspend a task for the given time.
*/
void OS_Delay(unsigned NumTicks) {

  [OS specific code ...]

  SEGGER_SYSVIEW_OnTaskStopReady(OS_Global.pCurrentTask, OS_CAUSE_WAITING);
}
```

```
/*********************************************************************
*
*       OS_HandleTick()
*
*  Function description
*    OS System Tick handler.
*/
int OS_HandleTick(void) {
  int TaskReady = 0;   // Pseudo variable indicating a task is ready

  [OS specific code ...]

  if (TaskReady) {
    SEGGER_SYSVIEW_OnTaskStartReady((unsigned)pTask);
  }
}

/*********************************************************************
*
*       OS_Switch()
*
*  Function description
*    Switch to the next ready task or go to idle.
*/
void OS_Switch(void) {

  [OS specific code ...]

  //
  // If a task is activated
  //
  SEGGER_SYSVIEW_OnTaskStartExec((unsigned)pTask);
  //
  // Else no task activated, go into idle state
  //
  SEGGER_SYSVIEW_OnIdle()
}

/*********************************************************************
*
*       OS_EnterInterrupt()
*
*  Function description
*    Inform the OS about start of interrupt execution.
*/
void OS_EnterInterrupt(void) {

  [OS specific code ...]

  SEGGER_SYSVIEW_RecordEnterISR();
}

/*********************************************************************
*
*       OS_ExitInterrupt()
*
*  Function description
*    Inform the OS about end of interrupt execution and switch to
*    Scheduler if necessary.
*/
void OS_ExitInterrupt(void) {

  [OS specific code ...]
  //
  // If the interrupt will switch to the Scheduler
  //
  SEGGER_SYSVIEW_RecordExitISRToScheduler();
  //
  // Otherwise
  //
  SEGGER_SYSVIEW_RecordExitISR();
}
```

Chapter 3

SEGGER SystemView API functions

The following functions can be used to integrate SYSTEMVIEW into an OS.

Function	Description
Control and initialization	
SEGGER_SYSVIEW_Init	
SEGGER_SYSVIEW_SendTaskList	
SEGGER_SYSVIEW_SendTaskInfo	
SEGGER_SYSVIEW_SendSysDesc	
Event recording	
SEGGER_SYSVIEW_RecordVoid	
SEGGER_SYSVIEW_RecordU32	
SEGGER_SYSVIEW_RecordU32x2	
SEGGER_SYSVIEW_RecordU32x3	
SEGGER_SYSVIEW_RecordSystime	
SEGGER_SYSVIEW_RecordEnterISR	
SEGGER_SYSVIEW_RecordExitISR	
SEGGER_SYSVIEW_RecordExitISRToSched-uler	
SEGGER_SYSVIEW_OnIdle	
SEGGER_SYSVIEW_OnTaskCreate	
SEGGER_SYSVIEW_OnTaskStartExec	
SEGGER_SYSVIEW_OnTaskStopExec	
SEGGER_SYSVIEW_OnTaskStartReady	
SEGGER_SYSVIEW_OnTaskStopReady	
SEGGER_SYSVIEW_OnUserStart	
SEGGER_SYSVIEW_OnUserStop	
SEGGER_SYSVIEW_NameResource	
SEGGER_SYSVIEW_SendPacket	
Event parameter encoding	

Function	Description
SEGGER_SYSVIEW_EncodeU32	
SEGGER_SYSVIEW_EncodeData	
SEGGER_SYSVIEW_EncodeString	
SEGGER_SYSVIEW_EncodeId	

3.0.1 SEGGER_SYSVIEW_EncodeData

3.0.2 SEGGER_SYSVIEW_EncodeId

3.0.3 SEGGER_SYSVIEW_EncodeString

3.0.4 SEGGER_SYSVIEW_EncodeU32

3.0.5 SEGGER_SYSVIEW_Init

3.0.6 SEGGER_SYSVIEW_NameResource

3.0.7 SEGGER_SYSVIEW_OnIdle

3.0.8 SEGGER_SYSVIEW_OnTaskCreate

3.0.9 SEGGER_SYSVIEW_OnTaskStartExec

3.0.10 SEGGER_SYSVIEW_OnTaskStartReady

3.0.11 SEGGER_SYSVIEW_OnTaskStopExec

3.0.12 SEGGER_SYSVIEW_OnTaskStopReady

3.0.13 SEGGER_SYSVIEW_RecordEnterISR

3.0.14 SEGGER_SYSVIEW_RecordExitISR

3.0.15 SEGGER_SYSVIEW_RecordExitISRToScheduler

3.0.16 SEGGER_SYSVIEW_RecordSystime

3.0.17 SEGGER_SYSVIEW_RecordU32

3.0.18 SEGGER_SYSVIEW_RecordU32x2

3.0.19 SEGGER_SYSVIEW_RecordU32x3

3.0.20 SEGGER_SYSVIEW_RecordVoid

3.0.21 SEGGER_SYSVIEW_SendPacket

3.0.22 SEGGER_SYSVIEW_SendSysDesc

3.0.23 SEGGER_SYSVIEW_SendTaskInfo