IAR Application Note #52086

Integrating a visualSTATE application with a Real-Time Operating System (RTOS)

The information in this document is based on version 5.0.4 of the IAR visualSTATE software. It may also apply to subsequent versions of IAR visualSTATE.

SUMMARY

This application note describes how to structure a visualSTATE application program using multiple visualSTATE Systems and a Real-Time Operating System (RTOS). Different ways of organizing multiple visualSTATE Systems in multiple tasks and how to structure the handling of events will be shown.

An example of a visualSTATE application using IAR Embedded Workbench for Mitsubishi M16C/6x and Segger embOS for Mitsubishi M16C will be used to show the application program structure.

KEYWORDS

Expert API, multiple visualSTATE Systems, RTOS, task, event queue, main loop

The problem to be solved

In some control logic applications with state machine based behavior, it can be useful with multiple and independently running state machines to take care of separate control tasks. Using multiple visualSTATE systems and one visualSTATE Expert API can handle this.

In many real-time embedded applications the various control tasks will be handled by a Real-Time Operating System (RTOS) for scheduling, prioritizing, and handling intercommunication.

The problem to be solved by this application note is how to structure the application program, the visualSTATE main loops, and the tasks to handle more than one visualSTATE system using one Expert API.

Note: Interprocess communication and synchronization using RTOS features (semaphores, mailboxes, etc.) will not be covered by this application note.

Solution

Designing and generating code for multiple visualSTATE systems

For a description of how to create multiple visualSTATE systems, and of the Expert API code generation, see *IAR visualSTATE User Guide*.

Organizing multiple visualSTATE systems in multiple tasks

When an application using multiple visualSTATE systems in multiple tasks is to be organized, it is a question of how many systems should be controlled by each task. In the two figures below, two different situations are illustrated.

Figure 1 shows an organization where each of two tasks is controlling one visualSTATE system. VS_System_1 and VS_System_2 mark the generated code for each visualSTATE system, and they are controlled by the two tasks named VS_Task_1 and VS_Task_2 respectively. Each of the two tasks is controlling its visualSTATE system via the Expert API library functions, and there is no direct access from the task to the code generated; all access is made via the Expert API.

The handling of events is done by a separate task named Input_Driver_Task. This task handles the conversion of inputs from the environment to events, and it makes sure that each event is added to the proper event queue dedicated to a specific visualSTATE system.

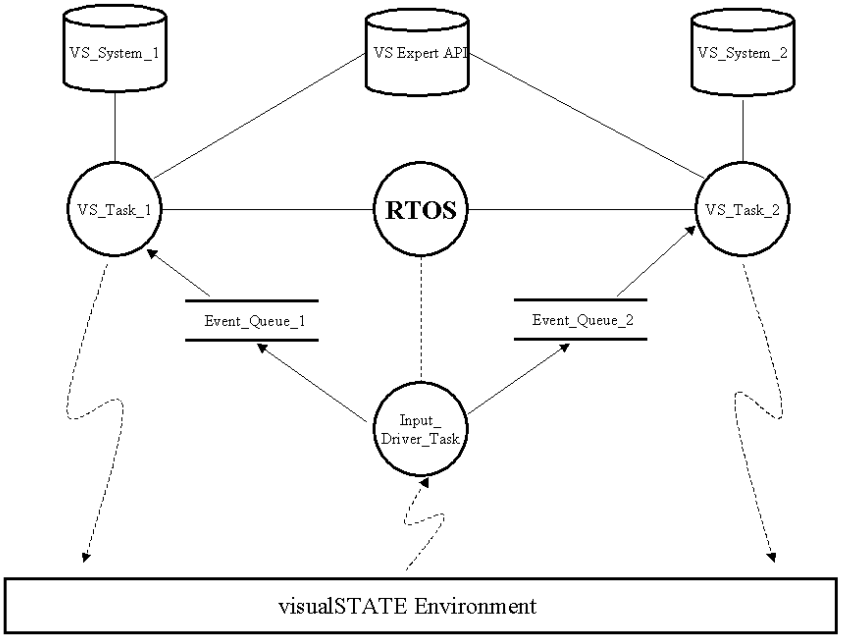

Figure 1: An implementation with 2 visualSTATE systems in separate tasks

Figure 2 shows a slightly different organization, where two tasks handle three visualSTATE systems. One of the tasks (VS_Task_2) is now handling two systems (VS_System_2 and VS_System_3) using two event queues, each dedicated for one system only.

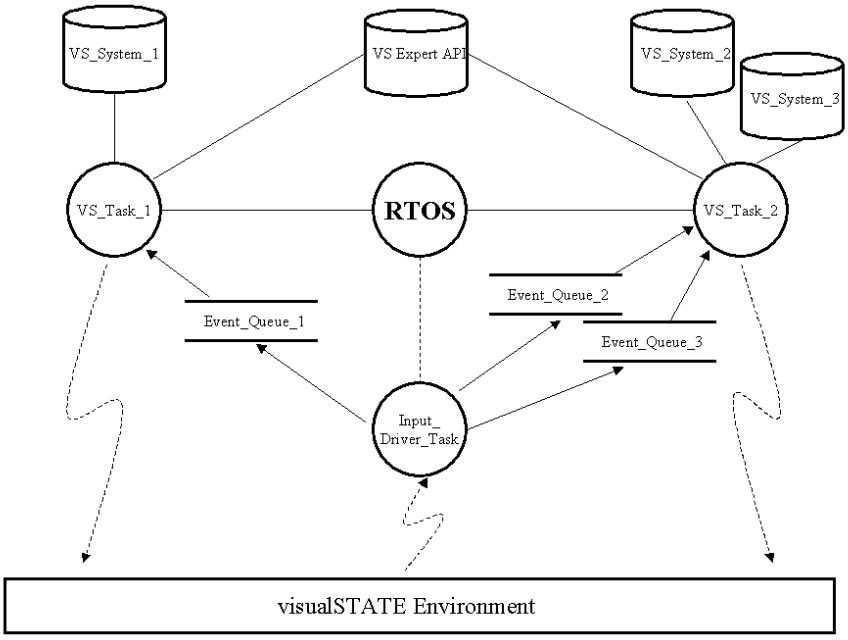

Figure 2: An implementation of three visualSTATE systems; VS_Task_2 now handles two systems

These two organizations of multiple visualSTATE systems in multiple tasks differ in how many systems are controlled by a task. How to structure the two main loops in the two different tasks will be shown in the next section.

Creating multiple tasks to control one or more visualSTATE systems

In this section, the following steps will be shown by code snippets to structure an application program:

1. Adding the generated C files to the compiler project
2. Including the generated C header files
3. Creating multiple tasks using an RTOS
4. Handling of events
5. Adding code to thevisualSTATE main

Step 1: Adding the generated C files to the compiler project

For a visualSTATE project with two systems the following five generated files and the Expert API files must be added to the compiler project:

- <VSProject Name>gextvar.c
  - in the example for this app. note: VS_System_Task1Data.c
- <VS_System_1>.c (VS_System_Task1.c)
- <VS_System_1>Data.c (VS_System_Task1Data.c)
- <VS_System_2>.c (VS_System_Task2.c)
- <VS_System_2>Data.c (VS_System_Task2Data.c)
- SMPMain.c

Step 2: Including the generated C header files

The following C header files generated by visualSTATE and for the Expert API must be included in the compiler project for the program to be able to access the event numbers, system context, etc.:

- <VS_System_1>.h (VS_System_Task1.h)
- <VS_System_1>Data.h (VS_System_Task1Data.h)
- <VS_System_2>.h (VS_System_Task2.h)
- <VS_System_2>Data.h (VS_System_Task2Data.h)
- <VSProject Name>Constant.h (RTOS_demo Constant.h)
- <VSProject Name>gevent.h (RTOS_demogevent.h)
- <VSProject Name>gextvar.h (RTOS_demogextvar.h)
- SEMLibE.h

The code snippet for including these files and the C header file for the Event Handler (simpleEventHandler-2) module could look like this:

```
#include "simpleEventHandler-2.h"


#include "SEMLibE.h"


#include "VS_System_Task1.h"
#include "VS_System_Task1Data.h"
#include "VS_System_Task1Action.h"
#include "VS_System_Task2.h"
#include "VS_System_Task2Data.h"
#include "VS_System_Task2Action.h"

#include "VS_RTOS_demoConstant.h"
#include "VS_RTOS_demogevent.h"
#include "VS_RTOS_demogextvar.h"
```

Step 3: Creating multiple tasks using an RTOS

In the main file for the compiler project, the tasks can be created as in this code snippet:

```
#include "VSmain.h"
#include "RTOS.H"

// Stack Space
OS_STACKPTR int Stack1[128];
OS_STACKPTR int Stack2[128];

// Task Control Blocks
OS_TASK TCB1;
OS_TASK TCB2;

void Task1(void)
{
   VS_MainLoop_Task1();
}

void Task2(void)
{
   VS_MainLoop_Task2();
}

int main(void)
{
   // Initialize the device by means of Input/Output, interrupts etc.
   InitDevice();

   // Initialize Segger embOS
   OS_InitKern();
   // Initialize Hardware for embOS
   OS_InitHW();

   // Here we create the two RTOS tasks controlling one VS system each
   OS_CREATETASK(&TCB1, "Task1", Task1, 100, Stack1);
   OS_CREATETASK(&TCB2, "Task2", Task2, 100, Stack2);

   // Start the multitasking
   OS_Start();

   // This code is never reached.
   return 0;
}
```

Step 4: Handling of events

To handle inputs from the environment, conversion to events, and adding to the event queues, a separate task could be created to handle this, or, as shown in the code snippet, a number of HW-related interrupt service routines (ISR) can be created. As shown in the code snippet, the events will be added to either or both event queues, depending on whether the events are defined at the project level or the system level.

In the code example, it can be seen that one of the three events (e_Button_INT0) is defined at the project level. It is also declared in both systems, and may thus be used in both systems. To handle this, the event handler function SEQ_AddEvent is called twice.

```
void SWITCH_init( void )
{
    INT0IC = 0x6;                   // Enable INT0 interrupt
    INT1IC = 0x6;                   // Enable INT1 interrupt
    INT2IC = 0x6;                   // Enable INT2 interrupt
}

void InitDevice( void )
{
    SWITCH_init();
    enable_interrupt();
}

interrupt [29*4] void INT0_interrupt( void )
{
    // Add event to the both event queues
    // - to be retrieved by visualSTATE main loop
    SEQ_AddEvent( 0, e_Button_INTO );
    SEQ_AddEvent( 1, e_Button_INTO );
}

interrupt [30*4] void INT1_interrupt( void )
{
    // Add event to the event queue for visualSTATE system "VS_System_Task1"
    // - to be retrieved by visualSTATE main loop
    SEQ_AddEvent( 0, e_Button_INT1 );
}

interrupt [31*4] void INT2_interrupt( void )
{
    // Add event to the event queue for visualSTATE system "VS_System_Task2"
    // - to be retrieved by visualSTATE main loop
    SEQ_AddEvent( 1, e_Button_INT2 );
}
```

Step 5: Adding code to the visualSTATE main loop

For each of the tasks controlling one visualSTATE system, the following code for the main loop must be added:

Note that the called visualSTATE initialization function is named <VS_System Name>SMP_InitAll, and that the called deduct function is named <VS_System Name>SMP_Deduct. This deduct function is code-generated if variable double-buffering is needed, and then this function must be called.

```
void VS_MainLoop_Task1( void )
{
   // Define completion code variable.
   unsigned char cc;

   // Define pointer to VS system context
   SEM_CONTEXT *pSEMContext;

   // Define action expression variable.
   SEM_ACTION_EXPRESSION_TYPE actionExpressNo;

   // Define and initialize. In this case the reset event is SE_RESET.
   SEM_EVENT_TYPE eventNo = SE_RESET;

   // Initialize the VS System.
   VS_System_Task1SMP_InitAll(&pSEMContext);

   // Initialize event queue
   SEQ_Initialize( 0 );

   SEQ_AddEvent( 0, SE_RESET );

   // mainloop
   while (1)
   {
      if (SEQ_RetrieveEvent( 0, &eventNo) != UCC_QUEUE_EMPTY)
      {
         // Deduct the event
         if ((cc = VS_System_Task1SMP_Deduct( pSEMContext, eventNo )) != SES_OKAY)
            HandleError_Task1(cc);

         // Get resulting action expressions and execute them
         while ((cc = SMP_GetOutput( pSEMContext, &actionExpressNo )) == SES_FOUND)
            SMP_TableAction( pSEMContext, VS_System_Task1VSAction, actionExpressNo );

         if (cc != SES_OKAY)
            HandleError_Task1(cc);

         // Change the next state vector
         if ((cc = SMP_NextState( pSEMContext )) != SES_OKAY)
            HandleError_Task1(cc);

      }
  } // End of mainloop
}
```

For each task controlling more than one visualSTATE system, the main loop program could look like the code snippet below. The code represents a task controlling two visualSTATE systems, each getting events via an event queue indexed by 1 and 2, and the names of the two systems are VS_System_Task1 and VS_System_Task2.

```
void VS_MainLoop_Task1( void )
{
  /*
  Initializing as illustrated in previous example
  Remember to initialize both systems
  */
  // mainloop
  while (1)
  {
      // Check event queue and deduct for System 1
      if (SEQ_RetrieveEvent( 1, &eventNo) != UCC_QUEUE_EMPTY)
      {
         // Deduct the event
         if ((cc = VS_System_Task1SMP_Deduct( pSEMContext1, eventNo )) != SES_OKAY)
            HandleError_Task1(cc);

         // Get resulting action expressions and execute them
         while ((cc = SMP_GetOutput( pSEMContext1, &actionExpressNo )) == SES_FOUND)
           SMP_TableAction( pSEMContext1, VS_System_Task1VSAction, actionExpressNo );

         if (cc != SES_OKAY)
            HandleError_Task1(cc);

         // Change the next state vector
         if ((cc = SMP_NextState( pSEMContext1 )) != SES_OKAY)
            HandleError_Task1(cc);
      }

      // Check event queue and deduct for System 2
      if (SEQ_RetrieveEvent( 2, &eventNo) != UCC_QUEUE_EMPTY)
      {
         // Deduct the event
         if ((cc = VS_System_Task2SMP_Deduct( pSEMContext2, eventNo )) != SES_OKAY)
            HandleError_Task1(cc);

         // Get resulting action expressions and execute them
         while ((cc = SMP_GetOutput( pSEMContext2, &actionExpressNo )) == SES_FOUND)
           SMP_TableAction( pSEMContext2, VS_System_Task1VSAction, actionExpressNo );

         if (cc != SES_OKAY)
            HandleError_Task1(cc);

         // Change the next state vector
         if ((cc = SMP_NextState( pSEMContext2 )) != SES_OKAY)
            HandleError_Task1(cc);

      }
  } // End of mainloop
}
```

Conclusions

As this application note shows it is quite easy to integrate a visualSTATE application with an RTOS. Furthermore, due to the ANSI C code generation of visualSTATE systems, the implementation of the target application is very flexible, and thus allows the user to design the application to his/her needs.

Contact information

SWEDEN: IAR Systems AB
P.O. Box 23051, S-750 23 Uppsala
Tel: +46 18 16 78 00 / Fax: +46 18 16 78 38
Email: info@iar.se

GERMANY: IAR Systems AG
Posthalterring 5, D-85599 Parsdorf
Tel: +49 89 88 98 90 80 / Fax: +49 89 88 98 90 81
Email: info@iar.de

Japan: IAR Systems K.K.
1-22-17 Fuji-building 26,
Hyakunin-cho, Shinjuku-ku
Tokyo 169 0073
Tel: +81 3 5337 6436 / Fax: +81 3 5337 6130
Email: info@iarsys.co.jp

DENMARK: IAR Systems A/S
Lykkesholms Allé 100, DK-8260 Viby J
Tel: +45 87 34 11 00 / Fax: +87 34 11 90
Email: info@iar.dk

USA: IAR Systems US HQ - West Coast
Century Plaza, Foster City, CA 94404
Tel: +1 650 287-4250 / Fax: +1 650 287-4253
Email: info@iar.com

USA: IAR Systems - East Coast
2 Mount Royal, Marlborough, MA 01752
Tel: +1 508 485-2692 / Fax: +1 508 485-9126
Email: info@iar.com

UK: IAR Systems Ltd
Gainsborough Business Centre,
Hamilton House, Mabledon Place,
Euston, London WC1H 9BB
Tel: +44 207 554 85 85 / Fax: +44 207 554 85 86
Email: info@iarsys.co.uk

February 2003.