RTOS 4 Documentation

Real Time Operating System Design

Hekar Khani, Dan Evans, Samuel Lewis, Adrian Hyde
December 14, 2010

Project

Directory Layout

- /RTOS4 - Root directory for project
    - /milestone4 - Contains the Visual 68k simulation code
    - /RTOS4 - Static library project and solution file
    - /RTOS4_Test - Test application project

Build Files

milestone4/globals.c -This file houses the operating system's global variables
milestone4/globals.h - This file houses the operating system's global variables
milestone4/osinit.c - This file provides RTOS initialization support from the C programming perspective
milestone4/rtos.h -Global declarations specific to the OS
milestone4/service.c -This file will implement support for OS service calling. According to requirements, we must support Basic Atari System Services (BASS) as well as Graphical Atari System Services (GASS)
milestone4/shared.h - Public service call definitions
milestone4/stack.c - This file reserves user accessable memory for application stacks. Needed for OS initialization
milestone4/stdlib.h -Standard library defintions
milestone4/support.c -Additional C based support routines for RTOS
milestone4/task1.c - GetGlobalDataAddress test application
milestone4/task2.c - GetSystemTickCount test application
milestone4/timer.c - Ipl6 timer handler

* Note: although, the files below have *.cpp extensions, this is merely to allow linking with C++ applcations and does not mean they're *.cpp files.

RTOS4/rtos.h -Global declarations specific to the OS
RTOS4/service.cpp -This file will implement support for OS service calling. According to requirements, we must support Basic Atari System Services (BASS) as well as Graphical Atari System Services (GASS)

RTOS4/globals.cpp -This file houses the operating system's global variables
RTOS4/globals.h - This file houses the operating system's global variables
RTOS4/osinit.cpp - This file provides RTOS initialization support from the C programming perspective
RTOS4/stdlib.h -Standard library defintions
RTOS4/support.cpp - Additional C based support routines for RTOS
RTOS4/task1.cpp - Mock application
RTOS4/task2.cpp - Mock application

RTOS4_Test/draw.cpp - Supporting functions for drawing
RTOS4_Test/draw.h - Supporting functions for drawing
RTOS4_Test/global_const.cpp - Globals, constants and definitions
RTOS4_Test/global_const.h - Globals, constants and definitions
RTOS4_Test/main.cpp - Mainline of the application. Most of the work is done here

Building

The RTOS 4 milestone is split into two projects. One is built for the Visual 68k simulator, the other is built for simulation and testing on Windows.

Visual 68k

The compiling of the milestone4 for the Visual 68k requires the installation of:

- The 68000 Cross Development System. This is freely downloadable
- The Visual 68k emulator (for 68000 simulation). This is also freely downloadable

The 68000 Cross Development System is a 16bit application and hence requires an operating system that is backwards compatible. If you are running at 64bit version of Windows, you will need to run the development system under DosBox. This is outside the scope of this tutorial.

To use the compiler, one has to open the command prompt (Winkey + R and type cmd). Enter the directory where the 68000 Cross Development System was installed, then execute the setpath.bat batch file. Afterwards, using that same command prompt, enter the directory of the RTOS4/milestone4 project and type make. The project should successfully make.

Visual Studio 2010

The Windows portion of the RTOS 4 milestone is comprised of two applications. One is a statically linked library and is direct port the needed RTOS source. Though the files have their extensions renamed to *.cpp an effort is made to not perform non-C compatible operations. The renaming to *.cpp is merely done to ensure linking with the Windows based test application. The test application produces a graphical executable to perform test functions on the RTOS code.

To build the solution it is rather simple. One merely requires to open the RTOS4.sln found in

the RTOS4/ directory and click Build->Build Solution from the menu bar. The project files were made under Visual Studio 2010.

TRAPS

1. BASS

Enter these into RegD0 call Trap #1

ie.

```
MOVE.L      #3, D0
TRAP        #1
```

```
/*!
 * Get a keyboard character
 */
#define BASS_GET_KEYBOARD           3

/*!
 * Is there available information in keyboard buffer
 */
#define BASS_KEYBOARD_STATUS        4

/*!
 * Get a debug character
 */
#define BASS_GET_DEBUG              5

/*!
 * Is there available information in debug buffer
 */
#define BASS_DEBUG_IN_STATUS        6

/*!
 * Write provided character to debug port
 * D1 - Character to write
 */
#define BASS_WRITE_DEBUG            7

/*!
 * Is the debug port busy
 */
#define BASS_DEBUG_BUSY             8
```

```
/*!
 * Retreive system tick count
 */
#define BASS_TICK_COUNT                 9

/*!
 * Retrieve task global memory address
 */
#define BASS_GLOBAL_ADDRESS             10

/*!
 * Get tick count formated into time
 * D1 - Time struct
 */
#define BASS_GETCLOCKTIME               11
```

Examples

InputKeyboardCharacter

```
/* Read in a string from the keyboard. */
short input[5];
int i;
for( i = 0; i < 5; ++i )
{
   input[i] = InputKeyboardCharacter();
}
```

InputDebugCharacter

```
/* Read in a string from the debugging device. */
short input[5];
int i;
for( i = 0; i < 5; ++i )
{
   input[i] = InputDebugCharacter();
}
```

OutputDebugCharacter

```
/* Output a string to the debugging device. */
short* output = "debug output"
```

```
int i = 0;
for( i = 0; output[i] != '\0'; ++i )
{
   OutputDebugCharacter( output[i] );
}
```

GetClockTime

```
struct systemtime time;
GetClockTime( &time );
```

Services

InputKeyboardCharacter

Description

Read the next available keyboard character from the system's internal keyboard buffer. This call blocks a process until a key is available.

Parameters

None

Return

Next available character as short.

IsKeyboardCharacterAvailable

Description

Checks if there are any characters in the keyboard buffer

Parameters

None

Return

TRUE - 1 or more characters are in the buffer
FALSE - No characters are in the buffer

InputDebugCharacter

Description

Read the next available character from the system's debugger buffer. This call blocks a process

until a character is available.

Parameters
None

Return
Next available character as short.

IsDebugCharacterAvailable

Description
Checks if there are any characters currently in the debug input bugger

Parameters
None

Return
TRUE - 1 or more characters are in the buffer
FALSE - 0 characters are in the buffer

OutputDebugCharacter

Description
Output a short to the system’s current debugging console.

Parameters
Short - WORD to output to debugger through serial port

Return
None

IsDebugPortBusy

Description
Checks if system's debugging device port is busy and returns TRUE or FALSE depending on the status.

Parameters
None

Return
TRUE - if system's debugging device port is busy
FALSE - otherwise

GetSystemTickCount

Description
Retrieves the number of timer interrupts that have occurred since the system booted

Parameters
None

Return
Unsigned Long - number of timer interrupts

GetGlobalDataAddress

Description
Retrieves a pointer to the calling tasks global data

Parameters
None

Return
Unsigned Long - address to the tasks global data

GetClockTime

Description
Returns the number of seconds since boot formatted into hours, minutes, and seconds

Parameters
Pointer to a struct systemtime

Return
None

GetSystemTickCount

Description
Return the number of ticks since the operating system has booted up. Each tick is specified by the scheduler's interruption cycle

Parameters

None

Return

Current operating system tick count as unsigned long