

How To Port MicroC/OS-II from Micrium on ST100 DSP?

This document describes how to port and run MicroC/OS-II on the ST122/ST140 processors.

MicroC/OS-II is demonstrated using the ST100-CEB-TOM2 board for the ST122 processor and using the ST140 CBS simulator for the ST140. As the ST100 DSP's are powerful CPU's, MicroC/OS-II is configured to provide full services and high performances.

Table of Contents

- 1. Assumptions
- 2. References
 - 2.1 Book
 - 2.2 Contacts
- 3. Licensing
- 4. Installation
 - 4.1 Files
 - 4.2 Configuration
- 5. Load the Sample Code into the ST122
- 6. How the Sample Code Works
- 7. MicroC/OS-II Port for ST122/ST140
 - 7.1 Directory and files
 - 7.2 os_cpu.h
 - 7.2.1. Data types
 - 7.2.2 OS_ENTER_CRITICAL and OS_EXIT_CRITICAL
 - 7.2.3 Stack growth
 - 7.2.4 Task Level Context Switch
 - 7.2.5 Function Prototypes
 - 7.3 os_cpu.c
 - 7.3.1 Hardware Loops
 - 7.3.2 OS_MemClr() and OS_MemCopy()
 - 7.3.3 OSTaskStkInit()
 - 7.3.4 MicroC/OS-II Task' stack
 - 7.4 os_cpu_a.s
 - 7.4.1 OSCPUsaveSR() and OSCPUrestoreSR()
 - 7.4.2 OSStartHighRdy()
 - 7.4.3 OSCtxSw()
 - 7.4.4 OSTickISR()
 - 7.4.5 OSIntCtxSw()
 - 7.4.6 Task_switch
 - 7.4.7 Pseudo-code for ALL ISRs
- 8. New Project Using MicroC/OS-II ST100 Port
 - 8.1 Standard Project Makefile
 - 8.2 Kernel library port specific Makefile
- 9. ST100 Port Notes
 - 9.1 Kernel library startup file *crt0.s*
 - 9.2 Interrupt Latency
 - 9.3 Stacks in ST122 interleaved memory

1. Assumptions

ST100 DSP Toolset 2.2.0: It is assumed that you have installed the ST100 DSP Toolset version 2.2.0 or higher

version for Solaris or Windows. It is also assumed that you are familiar with the ST100 DSP Toolset.

MicroC/OS-II: We assumed that you are familiar with MicroC/OS-II and that you have the book written by Mr. Jean J. Labrosse: *MicroC/OS-II, The Real-Time Kernel, 2nd Edition* (see References section).

2. References

2.1 Book

MicroC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-5782-0103-9

2.2 Contacts

CMP Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
USA
+1 785 841 1631
+1 785 841 2624 (FAX)
WEB: <http://www.rdbooks.com>
e-mail: rdorders@rdbooks.com

Micrium, Inc.
949 Crestview Circle
Weston, FL 33327
USA
+1 954 217 2036
+1 954 217 2037 (FAX)
e-mail: Jean.Labrosse@Micrium.com

3. Licensing

MicroC/OS-II source and object code can be used by accredited Colleges and Universities without requiring a license, as long as there is no commercial application involved. In other words, no licensing is required if MicroC/OS-II is used for educational use.

You need to obtain an '*Object Code Distribution License*' to embed MicroC/OS-II in a product that is sold with the intent to make a profit or if the product is not used for education or 'peaceful' research. For additional details, contact Micrium at Licensing@Micrium.com or by calling Micrium (see *Contacts*, above).

4. Installation

The Micrium, Inc. application note from Jean J. Labrosse AN-2002 defines a new directory structure used by Micrium for storing MicroC/OS-II ports (2.70 and higher).

The directory structure is the following one:

```

\Micrium
  \Software
    \uCOS-II
      \Doc                Release notes and manuals
      \Ports              Port Family Directories
        \st100            ST100 ports directory
          \doc             Documentation
          \st122           ST122 port
            \ex1           ST122 example
          \st140           ST140 port
            \ex1           ST140 example
        \Source           Processor independent source

```

The directory structure described in the application note AN-2002 suggests adding a directory level below 'st100' for port mode (optimized for speed/size) and one for compiler version. We do not follow this recommendation, as

it does not match our needs.

4.1 Files

Get the source files of MicroC/OS-II ports (2.70 and higher) to have the generic code of the kernel. Files are shown in **bold**.

```

\Micrium
  \Software
    \uCOS-II
      \DOC
        WhatsNewSince-V200.PDF
        ReleaseNotes.PDF
        QuickRefChart-Color.PDF
        uCOS-II-CfgMan.PDF
        uCOS-II-RefMan.PDF
        README.TXT
        TaskAssignmentWorksheet.PDF
        TaskAssignmentWorksheet.XLS
      \Source
        os_cfg_r.h
        os_core.c
        os_dbg_r.c
        os_flag.c
        os_mbox.c
        os_mem.c
        os_mutex.c
        os_q.c
        os_sem.c
        os_task.c
        os_time.c
        ucos_ii.c
        ucos_ii.h

```

The code coming with this application note is found in the file [uCOSII-st100.tar.gz](#). You simply ungzip/untar this file in the directory `Micrium/Software/uCOS-II/`. Once unzipped, you should find the following directory tree and files shown below. Files are shown in **bold**.

```

SLA.pdf
\ports
  \st100
    \doc
      Howto-Port-MicroCOS-II.htm
    \Images
      D0001.gif
      D0002.jpg
    \s122
      Makefile
      crt0.s
      os_cpu.h
      os_cpu_a.s
      os_cpu_c.c
      os_hkih.c
      os_hkihe.c
      os_hktcb.c
      os_hktch.c
      os_hktdh.c
      os_hktih.c
      os_hktsh.c
      os_hktth.c
      os_hktwh.c
    \exl
      includes.h
      main.c
      Makefile
      my_it.s
      os_cfg.h
  \st140

```

```
Makefile
crt0.s
os_cpu.h
os_cpu_a.s
os_cpu_c.c
os_hkih.c
os_hkihe.c
os_hktcb.c
os_hktch.c
os_hktdh.c
os_hktih.c
os_hktsh.c
os_hktth.c
os_hktwh.c
\exl
includes.h
main.c
Makefile
my_it.s
os_cfg.h
```

- [SLA.pdf](#) is the Software License Agreement.
- [Ports](#) contains all the processor specific code (a.k.a. Ports) for MicroC/OS-II.
- [st100](#) contains all processors in the ST100 series.
- [st122](#) represents the ST100 series of processors specific port files and in this case, for the ST122 processor.
- [st140](#) represents the ST100 series of processors specific port files and in this case, for the ST140 processor.
- [Ex1](#). this directory contains the files used to build example #1

4.2 Configuration

Compilation

No configuration is required with the ST100 Toolset 2.2.0

Directory names

The ST100 port try to locate the generic kernel source files in a directory called [source](#). On case-sensitive system like Solaris or Linux, you may have to rename the directory created by the generic distribution from Micrium (Source directory).

Optimizations

- Two kernel functions, [OS_MemCopy\(\)](#) and [OS_MemClr\(\)](#), can be compiled using hardware loops for better performances. The ST122 and ST140 ports compile these functions if you comment the generic ones from the file:

```
Micrium/Software/uCOS-II/source/os_core.c
```

And uncomment the ones from files:

```
Micrium/Software/uCOS-II/ ports/st100/st122/os_cpu_c.c
Micrium/Software/uCOS-II/ ports/st100/st140/os_cpu_c.c.
```

- We remove some calls to software hooks in kernel sources. We do not deliver these source files.

5. Load the Sample Code into the ST122

Using a ST100 Toolset 2.2.0 bash shell command on Windows or a configured shell on Solaris, go in the [ports/st100/st122/ex1](#) directory.

Execute the following command:

```
make
```

(Note if the compiler fails to compile the [os/os_flags.o](#) file on UNIX, the kernel source files are probably in a

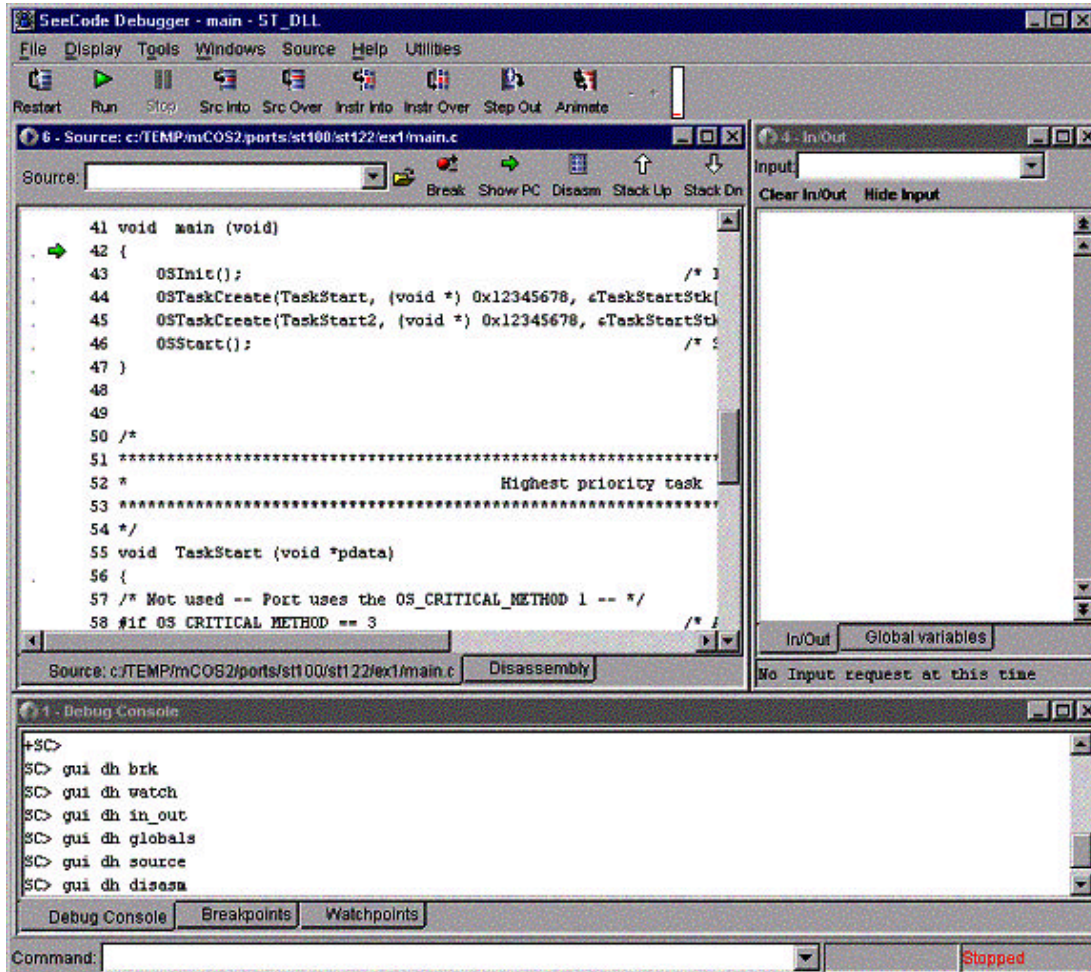
directory called `Source` instead of `source`).

The MicroC/OS-II kernel and the `main.c` files should compile.

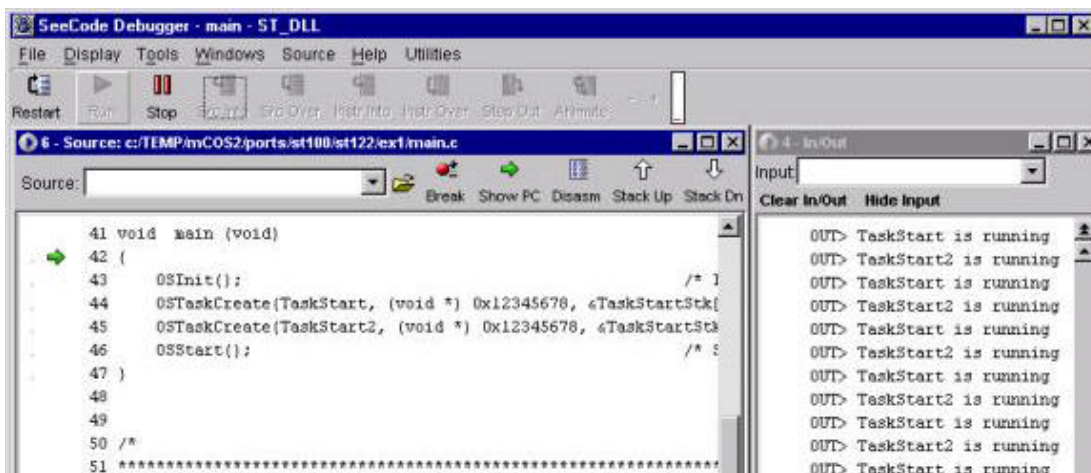
Start the ST100 debugger (*SeeCode™*) using the following command:

```
scst-OKN main
```

SeeCode should displays the following window:



You type on the Run button, the program starts and displays some outputs in the In/Out window



```

52 *           Highest priority task
53 *****
54 */
55 void TaskStart (void *pdata)
56 {
57 /* Not used -- Port uses the OS_CRITICAL_METHOD 1 -- */
58 #if OS_CRITICAL_METHOD == 3

```

```

OUT> TaskStart2 is running
OUT> TaskStart is running
OUT> TaskStart2 is running
OUT> TaskStart is running
OUT> TaskStart2 is running
OUT> TaskStart is running
OUT> TaskStart2 is running

```

Source: c:\TEMP\mCOS2\ports\st100\st122\ex1\main.c Disassembly

1 - Debug Console

```

SC> gui dh globals
SC> gui dh source
SC> gui dh disasm
SC> run
WARNING : PC=0x000025A4 : Hardware loop : POP of loop register results in undefined hardware loop machine sta
WARNING : PC=0x00002618 : Hardware loop : POP of loop register results in undefined hardware loop machine sta

```

Debug Console Breakpoints Watchpoints

Command: Run #

This example program simply switches from the task `TaskStart()` to the `TaskStart2()`.

6. How the Sample Code Works

The sample code consists in 2 tasks `TaskStart()` and `TaskStart2()`.

`TaskStart()` is the highest priority task. It displays a message `TaskStart() is running` and suspends.

`TaskStart2()` runs when `TaskStart()` is in suspended mode. It displays the message `TaskStart2() is running` and resumes `TaskStart()`. The two tasks are scheduled using the suspend/resume kernel functions.

7. MicroC/OS-II Port for ST122/ST140

The MicroC/OS-II port for ST122/ST140 assumes the ST100 Toolset 2.2.0.

However, you can certainly modify this port to work with other C/C++ compilers. In fact, the CPU instructions (i.e. the code) should be identical and all you would have to do is adapt the files to your compiler specifics (compiler/assembler/linker directives and options). We will describe some of these when we cover the contents of the different files.

We assume that you have MicroC/OS-II V2.70 or higher.

7.1 Directory and files

The software that goes with this application note is provided on the Micrium web site (connect to <http://www.Micrium.com> and then follow the Port link). When you unzip the file, the port files are found in the following directories:

[Micrium/Software/uCOS-II/ ports/st100/st122/](#) for ST122 core

[Micrium/Software/uCOS-II/ ports/st100/st140/](#) for ST140 core

Like all MicroC/OS-II ports, the source code for the port is found in the following three files:

```

os_cpu.h
os_cpu_a.s
os_cpu_c.c

```

7.2 os_cpu.h

`os_cpu.h` contains processor- and implementation-specific `#define`s constants macros, and typedefs.

7.2.1 Data types

```

typedef unsigned char BOOLEAN;

```

```

typedef unsigned char  INT8U;    /* Unsigned 8-bit quantity */
typedef signed  char  INT8S;    /* Signed 8-bit quantity */
typedef unsigned short INT16U;  /* Unsigned 16-bit quantity */
typedef signed  short INT16S;  /* Signed 16-bit quantity */
typedef unsigned int  INT32U;  /* Unsigned 32-bit quantity */
typedef signed  int   INT32S;  /* Signed 32-bit quantity */
typedef float        FP32;     /* Single precision floating point */
typedef double       FP64;     /* Double precision floating point */

typedef INT32U      OS_STK;     /* Each stack entry is 32-bit wide */
typedef INT32U      OS_CPU_SR; /* Define size of CPU status register (PSR) */

```

The first lines define the kernel standard type for compiler abstraction. The ST100 core uses a 32-bits wide stack. Each kernel stack uses OS_STK as its data type. The program status register (PSR) is 32-bits wide. The OS_CPU_SR data type is used when OS_CRITICAL_METHOD #3 is used. The ST100 port does not use this method.

7.2.2 OS_ENTER_CRITICAL and OS_EXIT_CRITICAL

```

#define OS_CRITICAL_METHOD 1

#if OS_CRITICAL_METHOD == 1
#define OS_ENTER_CRITICAL() asm(".if isgp16()\n\n\t.pregp32\n\tgp32md\n\tbfpsrl 0x1f, 0x1 f\n\n\t.pregp16\n\tgp16md\n\n\t.else\n\tbfpsrl 0x1f, 0x1f\n\n\t.endif")

#define OS_EXIT_CRITICAL() asm(".if isgp16()\n\n\t.pregp32\n\tgp32md\n\tbfpsrl 0x1f, 0x00\n\n\t.pregp16\n\tgp16md\n\n\t.else\n\tbfpsrl 0x1f, 0x0\n\n\t.endif")

#endif

```

MicroC/OS-II uses OS_ENTER_CRITICAL and OS_EXIT_CRITICAL to protect critical section of code. The book (MicroC/OS-II, The Real-Time Kernel) describes three methods to enable/disable interruptions. The ST100 port uses the first one (OS_CRITICAL_METHOD #1).

OS_ENTER_CRITICAL and OS_EXIT_CRITICAL use assembler instructions to modify the program status register (PSR) to change the interrupt mask level (PSR.IML).

The code can be used either in GP16 or GP32 mode (using 16-bits or 32-bits change mode instructions).

7.2.3 Stack growth

```

#define OS_STK_GROWTH 1 /* Stack grows from HIGH to LOW memory on ST100 */

```

The stack on the ST100 grows from high memory to low memory.

7.2.4 Task Level Context Switch

MicroC/OS-II uses OS_TASK_SW macro to perform a context switch.

The standard method to perform a context switch consists in raising a software trap.

The above code uses several compilation flags.

If `KERNEL_NO_TRAP` is not set, `OS_TASK_SW` executes a 'trap 0xF' instruction. A software trap is raised and the function `OSCtxSw()` from file `os_cpu_a.s` is executed (same address as `_swtrapF`).

The kernel calls the `OS_TASK_SW` macro in only one function. If you compile the kernel with the `KERNEL_NO_TRAP` flag, no software trap will be used and the code of the `OSCtxSw()` function will be included and executed directly. Few cycles are won during context switch. The code is quite complex as it supports several compilations flags.

If `KERNEL_SWITCH_HOOK` is set `OSCtxSw()` and the built-in version will call `OSTaskSwHook()` function. This compilation flag is not set by default (win few cycles during context switch).

`KERNEL_GP16` is flag set when the kernel is compiled in GP16 mode. `OSCtxSw()` and the built-in version must include some code to use GP32 instructions.

`KERNEL_SAFE_MODE` saves all registers on stack during context switches. The default mode saves and restores only the non-scratch registers.

The ST140 port uses quite the same definitions. It saves registers using `vpush/vpop` instructions.

```

#ifdef KERNEL_NO_TRAP
    /* Emulate a trap store instruction */

(1)
    #ifdef KERNEL_GP16
        #define OS_TASK_SW0() asm(".pregp32\n \
                                gp32md\n");
    #else /* !KERNEL_GP16 */
        #define OS_TASK_SW0()
    #endif /* !KERNEL_GP16 */

(2)
    #define OS_TASK_SW1() asm("makea p14, %abs16to31(OSCtxSwEnd)\n \
                             morea p14, %abs0to15(OSCtxSwEnd)\n \
                             saw @(p15-!4), p14\n \
                             scw @(p15-!4), PSR");

(3)
    #ifdef KERNEL_SAFE_MODE
        /* Push all registers */
        /* Code for ST122 core */
        #define OS_TASK_SW2() asm("push 0x1FFFFFF");
    #else /* !KERNEL_SAFE_MODE */
        /* Push non-scratch registers
        p15U-p14, p4-p11, LE0-LS0-LCOR-LC0, r4-r11, LK, GR
        push 0x179E79 save on stack that it is a non-
        interrupt context switch */
        /* Code for ST122 core */
        #define OS_TASK_SW2()
            asm("push p15, p14, p4-p11, L0, L1, L2, fr, r4-r11, LK, GR\n \
                more r0, 0\n \
                sdw @(p15-!4), r0\n");
    #endif /* !KERNEL_SAFE_MODE */
    /* Save the current task's stack pointer into the
    current task's OS_TCB
    OSTCBCur->OSTCBStkPtr = stack pointer
    As OSTCBStkPtr is the first field of OSTCBCur,
    we can use OSTCBCur address directly */

(4)
    #define OS_TASK_SW3() asm("makea p14, %abs16to31(OSTCBCur)\n \
                             morea p14, %abs0to15(OSTCBCur)\n \
                             law p14, @(p14 + 0)\n \
                             saw @(p14+0), SP\n");
    /* OSTCBCur = OSTCBHighRdy
    OSPrioCur = OSPrioHighRdy */

(5)
    #define OS_TASK_SW41() asm("makea p1, %abs16to31(OSTCBHighRdy)\n \
                              makea p2, %abs16to31(OSTCBCur)\n \
                              morea p1, %abs0to15(OSTCBHighRdy)\n \
                              morea p2, %abs0to15(OSTCBCur)\n \
                              ldw r14, @(p1 + 0)\n \

```

```

makea    p14, %abs16to31(OSPrioHighRdy)\n \
sdw      @(p2 + 0), r14\n \
morea    p14, %abs0to15(OSPrioHighRdy)\n \
makea    p2, %abs16to31(OSPrioCur)\n \
ldub     r14, @(p14 + 0)\n \
morea    p2, %abs0to15(OSPrioCur)\n \
sdb      @(p2 + 0), r14\n");

```

(6)

```

/* stack pointer = OSPrioHighRdy->OSTCBStkPtr
   OSPrioHighRdy->OSTCBStkPtr is already stored in
   p1 */
#define OS_TASK_SW42() asm("law p1, @(p1 + 0)\n \
                           law SP, @(p1+0)\n");

```

(7)

```

#ifdef KERNEL_SAFE_MODE
/* Code for ST122 core */
/* Restore all non-scratch registers from the new
   task's stack
   Execute a return from interrupt instruction */
#define OS_TASK_SW5() asm("pop 0x1FFF80\n \
                           law p3, @(p15+4)\n \
                           law p2, @(p15!+8)\n \
                           poprte 0x3F\n \
                           OSTxSwEnd:\n");

```

```

#else /* !KERNEL_SAFE_MODE */
/* get the type of context switch done */
/* Code for ST122 core */
#define OS_TASK_SW5()
asm("ldw r0, @(p15!+4)\n \
    cmpgth g0, r0, 0\n \
    g0?goto full_switch\n \
    poprte p15, p14, p4-p11, L0, L1, L2, fr, r4-r11, LK, GR\n \
    full_switch:\n \
    pop 0x1FFF80\n \
    law p3, @(p15+4)\n \
    law p2, @(p15!+8)\n \
    poprte 0x3F\n \
    OSTxSwEnd:\n");
#endif /* !KERNEL_SAFE_MODE */

```

(8)

```

#ifdef KERNEL_GP16
#define OS_TASK_SW6() asm("gp16md\n");
#else /* !KERNEL_GP16 */
#define OS_TASK_SW6()
#endif /* !KERNEL_GP16 */

```

```

/* call user definable OSTaskSwHook if needed */
#ifdef KERNEL_SWITCH_HOOK

```

(9)

```

#define OS_TASK_SW() OS_TASK_SW0(); \
                    OS_TASK_SW1(); \
                    OS_TASK_SW2(); \
                    OS_TASK_SW3(); \
                    OSTaskSwHook(); \
                    OS_TASK_SW41(); \
                    OS_TASK_SW42(); \
                    OS_TASK_SW5(); \
                    OS_TASK_SW6();
#else /* KERNEL_SWITCH_HOOK */

```

(10)

```

#define OS_TASK_SW() OS_TASK_SW0(); \
                    OS_TASK_SW1(); \
                    OS_TASK_SW2(); \
                    OS_TASK_SW3(); \
                    OS_TASK_SW41(); \
                    OS_TASK_SW42(); \
                    OS_TASK_SW5(); \
                    OS_TASK_SW6();

```

```

        #endif /* KERNEL_SWITCH_HOOK */

#else /* KERNEL_NO_TRAP */

(11)
        #define uCOS          0xF /* software trap vector used for context switch */

        #define OS_TASK_SW_STR_ASM(X) asm(".if isgp16()\n \
        .pregp32\n \
        gp32md\n \
        trap " # X "\n \
        gp16md\n \
        .else\n \
        trap " # X "\n \
        .endif")

        #define OS_TASK_SW_0(X) OS_TASK_SW_STR_ASM(X)
        #define OS_TASK_SW() OS_TASK_SW_0(uCOS)

#endif /* KERNEL_NO_TRAP */

```

1. Switch from GP16 to GP32 instruction mode if needed.
2. Save the address of OSCtxSwEnd into stack for PCnext.
Save PSR on stack.
3. Save all registers in safe mode or only the non-scratch.
4. Save the current task's stack pointer into the current task's OS_TCB.
5. Modify the current TCB and priority to the highest priority.
6. Change the stack pointer
7. Restore all registers if KERNEL_SAFE_MODE flag is set. In default mode, restore all registers (if the top of stack is 1) or only the non-scratch (if the top of stack is 0).
8. Switch from GP32 to GP16 instruction mode if needed.
9. Full definition of OS_TASK_SW macro calling the OSTaskSwHook() function (KERNEL_SWITCH_HOOK flag set).
10. Full definition of OS_TASK_SW macro avoiding call to OSTaskSwHook().
This optimization is non-standard to MicroC/OS-II book.
11. Use the software trap to perform the context switch.

7.2.5 Function Prototypes

```

#if OS_CRITICAL_METHOD == 3
    OS_CPU_SR OS_CPU_SaveSR(void);
    void OS_CPU_RestoreSR(OS_CPU_SR cpu_sr);
#endif

```

These function prototypes are associated with OS_CRITICAL_METHOD #3 described above. The ST100 port does not use it.

7.3 os_cpu_c.c

A MicroC/OS-II port requires that you write ten fairly simple C functions:

```

OSTaskStkInit() MUST be in os_cpu_c.c
OSInitHookBegin()
OSInitHookEnd()
OSTaskCreateHook()
OSTaskDelHook()
OSTaskIdleHook()
OSTaskStatHook()
OSTaskSwHook()
OSTCBInitHook()
OSTimeTickHook()

```

MicroC/OS-II requires all these functions to be declared but, OSTaskStkInit() actually needs to contain code to 'prepare' the stack frame of each task. OSTaskStkInit() is always placed in os_cpu_c.c along with the other nine

functions. However, the nine other functions can be declared in another file if `OS_CPU_HOOKS_EN` is set to 0 in `os_cfg.h`. In fact, for the ST100 port, these functions are located in separate files. Thus, if the user writes an implementation for one of these functions, the linker will use it. Otherwise, the default implementation will be taken from the kernel library.

```
OSInitHookBegin()    default implementation is declared in os_hkihbc.c
OSInitHookEnd()     default implementation is declared in os_hkihec.c
OSTaskCreateHook()  default implementation is declared in os_hktchc.c
OSTaskDelHook()     default implementation is declared in os_hktdhc.c
OSTaskIdleHook()    default implementation is declared in os_hktihec.c
OSTaskStatHook()    default implementation is declared in os_hktshc.c
OSTaskSwHook()      default implementation is declared in os_hktwhc.c
OSTCBInitHook()     default implementation is declared in os_hktcb.c
OSTimeTickHook()    default implementation is declared in os_hktthc.c
```

7.3.1 Hardware Loops

```
/* Hardware loop initialization addresses */
extern void* ____end0;
extern void* ____start0;
extern void* ____end1;
extern void* ____start1;
extern void* ____end2;
extern void* ____start2;
```

The task's stack must be initialized using default values for the processor registers. The ST100 DSP uses hardware loop registers. These registers must be initialized using safe addresses values (`____end0 ... ____start2`), which are defined into `crt0.s` startup file.

7.3.2 OS_MemClr() and OS_MemCopy()

`OS_MemClr()` and `OS_MemCopy()` are two kernel functions defined in `os_core.c` file.

The section **Configuration** explains that we move these functions into `os_cpu_c.c` for optimization using hardware loop mechanism.

7.3.3 OSTaskStkInit()

`OSTaskStkInit()` initializes the task's stack.

```
OS_STK *OSTaskStkInit (void (*task)(void *pd), void *pdata, OS_STK *ptos, INT16U opt)
{
    INT32U *stk;

    opt = opt; /* 'opt' is not used, prevent warning */
    stk = (INT32U*) ptos; /* Load stack pointer */
    OS_MemClr( (INT8U*) &stk[-64], 64*sizeof(INT32U));
    stk[0] = (INT32U) task; /* PCnext */
    stk[-1] = (INT32U) 0x10000014; /* PSR */
    /* PSR.PM 0 1 Privilege Mode. (1=USER)
       1 1 Reserved for future use.
       PSR.NMIE 2 1 NMI Enable. (1=enable)
       3 1 Reserved for future use.
       PSR.TE 4 1 Trap Enable. (1=enable)
       5 1 Reserved for future use.
       6 1 Reserved for future use.
       7 1 Reserved for future use.
       PSR.IML 12-8 5 Interrupt Mask Level.
       13 1 Reserved for future use.
       14 1 Illegal.
       15 1 Reserved for future use.
       PSR.EM 18-16 3 Processor Execution Mode.
       PSR.CL 23-19 5 Processor Current Level.
       24 1 Reserved for future use.
       25 1 Reserved for future use.
       26 1 Reserved for future use.
       27 1 Reserved for future use.
       PSR.LE 28 1 Loop Enable.
       PSR.IM 30-29 2 Processor Instruction
```

```

Mode.
31 1 Reserved for future use. */

/* -2 */ /* P15U Pointer registers */
/* -3 */ /* P14 */
/* -4 */ /* P13 */
/* -5 */ /* P12 */
/* -6 */ /* P11 */
/* -7 */ /* P10 */
/* -8 */ /* P9 */
/* -9 */ /* P8 */
/* -10 */ /* P7 */
/* -11 */ /* P6 */
/* -12 */ /* P5 */
/* -13 */ /* P4 */
/* -14 */ /* P3 */
/* -15 */ /* P2 */
/* -16 */ /* P1 */
stk[-17] = (INT32U) pdata; /* P0 */
/* -18 */ /* FR Flag registers */
stk[-19] = (INT32U) &____end2; /* LE2 Hardware loop 2 registers */
stk[-20] = (INT32U) &____start2; /* LS2 */
/* -21 */ /* LC2R */
/* -22 */ /* LC2 */
stk[-23] = (INT32U) &____end1; /* LE1 Hardware loop 1 registers */
stk[-24] = (INT32U) &____start1; /* LS1 */
/* -25 */ /* LC1R */
/* -26 */ /* LC1 */
stk[-27] = (INT32U) &____end0; /* LE0 Hardware loop 0 registers */
stk[-28] = (INT32U) &____start0; /* LS0 */
/* -29 */ /* LC0R */
/* -30 */ /* LC0 */
/* -31 */ /* R15-MSB 40-bits registers */
/* -32 */ /* R15-LSB */
/* -33 */ /* R14-MSB */
/* -34 */ /* R14-LSB */
/* -35 */ /* R13-MSB */
/* -36 */ /* R13-LSB */
/* -37 */ /* R12-MSB */
/* -38 */ /* R12-LSB */
/* -39 */ /* R11-MSB */
/* -40 */ /* R11-LSB */
/* -41 */ /* R10-MSB */
/* -42 */ /* R10-LSB */
/* -43 */ /* R9-MSB */
/* -44 */ /* R9-LSB */
/* -45 */ /* R8-MSB */
/* -46 */ /* R8-LSB */
/* -47 */ /* R7-MSB */
/* -48 */ /* R7-LSB */
/* -49 */ /* R6-MSB */
/* -50 */ /* R6-LSB */
/* -51 */ /* R5-MSB */
/* -52 */ /* R5-LSB */
/* -53 */ /* R4-MSB */
/* -54 */ /* R4-LSB */
/* -55 */ /* R3-MSB */
/* -56 */ /* R3-LSB */
/* -57 */ /* R2-MSB */
/* -58 */ /* R2-LSB */
/* -59 */ /* R1-MSB */
/* -60 */ /* R1-LSB */
/* -61 */ /* R0-MSB */
/* -62 */ /* R0-LSB */
stk[-63] = (INT32U) 0x80808080; /* GR */
/* -64 */ /* LK */

#ifdef KERNEL_SAFE_MODE
return ((OS_STK *)&stk[-64]);
#else /* !KERNEL_SAFE_MODE */
stk[-65] = (INT32U) 0x1; /* Type of context switch = 1 */
return ((OS_STK *)&stk[-65]);
#endif /* !KERNEL_SAFE_MODE */
}

```

The ST100 DSP has a lot of registers. The above function is dedicated to the ST122 core; the ST140 one initializes more registers but it uses the same template. The function initializes the stack to perform a pop of all registers.

- It initializes the stack to zero, because a lot of registers have a zero default value.
- The 2 first elements of the stack contain the task address and the PSR, they will be restored using the return from exception (RTE) instruction.
- stk[-17] (register P0) is initialized using the task parameter (EABI request).
- stk[-19] ... stk[-28] are initialized using the default safe hardware loop values.

If the kernel is compiled in the safe mode (KERNEL_SAFE_MODE flag), then all registers are saved on stack for all context switches.

If the kernel is compiled in default mode:

- All registers are saved during asynchronous task switch (interruption).
- nly non-scratch registers are saved during kernel switch (the current task calls a kernel function performing a context switch).

The kernel must save on stack if it saves all registers or only the non-scratch, so we pop this information (0 for non-scratch only, 1 for all registers) in stk[-65].

7.3.4 MicroC/OS-II Task' stack

The default stack frame for the ST122 and the ST140 ports is described below. The ST140 stack base address is 64-bit aligned.

ST122	ST122 safe mode	ST140	ST140 safe mode
Pcnext = task	Pcnext = task	Pcnext = task	Pcnext = task
PSR	PSR	PSR	PSR
P15U	P15U	PSR	PSR
P14	P14	P15U	P15U
P13	P13	P14	P14
P12	P12	P13	P13
P11	P11	P12	P12
P10	P10	P11	P11
P9	P9	P10	P10
P8	P8	P9	P9
P7	P7	P8	P8
P6	P6	P7	P7
P5	P5	P6	P6
P4	P4	P5	P5
P3	P3	P4	P4
P2	P2	P3	P3
P1	P1	P2	P2
P0 = pdata	P0 = pdata	P1	P1
FR	FR	P0 = pdata	P0 = pdata
LE2 = ____end2	LE2 = ____end2	K0.FR	K0.FR
LS2 = ____start2	LS2 = ____start2	K1.FR	K1.FR
LC2R	LC2R	LE2 = ____end2	LE2 = ____end2
LC2	LC2	LS2 = ____start2	LS2 = ____start2
LE1 = ____end1	LE1 = ____end1	LC2R	LC2R
LS1 = ____start1	LS1 = ____start1	LC2	LC2
LC1R	LC1R	LE1 = ____end1	LE1 = ____end1

LC1	LC1	LS1 = ____start1	LS1 = ____start1
LE0 = ____end0	LE0 = ____end0	LC1R	LC1R
LS0 = ____start0	LS0 = ____start0	LC1	LC1
LC0R	LC0R	LE0 = ____end0	LE0 = ____end0
LC0	LC0	LS0 = ____start0	LS0 = ____start0
R15-MSB	R15-MSB	LC0R	LC0R
R15-LSB	R15-LSB	LC0	LC0
R14-MSB	R14-MSB	K1.R15-MSB	K1.R15-MSB
R14-LSB	R14-LSB	K0.R15-MSB	K0.R15-MSB
R13-MSB	R13-MSB	K1.R15-LSB	K1.R15-LSB
R13-LSB	R13-LSB	K0.R15-LSB	K0.R15-LSB
R12-MSB	R12-MSB	K1.R14-MSB	K1.R14-MSB
R12-LSB	R12-LSB	K0.R14-MSB	K0.R14-MSB
R11-MSB	R11-MSB	K1.R14-LSB	K1.R14-LSB
R11-LSB	R11-LSB	K0.R14-LSB	K0.R14-LSB
R10-MSB	R10-MSB	K1.R13-MSB	K1.R13-MSB
R10-LSB	R10-LSB	K0.R13-MSB	K0.R13-MSB
R9-MSB	R9-MSB	K1.R13-LSB	K1.R13-LSB
R9-LSB	R9-LSB	K0.R13-LSB	K0.R13-LSB
R8-MSB	R8-MSB	K1.R12-MSB	K1.R12-MSB
R8-LSB	R8-LSB	K0.R12-MSB	K0.R12-MSB
R7-MSB	R7-MSB	K1.R12-LSB	K1.R12-LSB
R7-LSB	R7-LSB	K0.R12-LSB	K0.R12-LSB
R6-MSB	R6-MSB	K1.R11-MSB	K1.R11-MSB
R6-LSB	R6-LSB	K0.R11-MSB	K0.R11-MSB
R5-MSB	R5-MSB	K1.R11-LSB	K1.R11-LSB
R5-LSB	R5-LSB	K0.R11-LSB	K0.R11-LSB
R4-MSB	R4-MSB	K1.R10-MSB	K1.R10-MSB
R4-LSB	R4-LSB	K0.R10-MSB	K0.R10-MSB
R3-MSB	R3-MSB	K1.R10-LSB	K1.R10-LSB
R3-LSB	R3-LSB	K0.R10-LSB	K0.R10-LSB
R2-MSB	R2-MSB	K1.R9-MSB	K1.R9-MSB
R2-LSB	R2-LSB	K0.R9-MSB	K0.R9-MSB
R1-MSB	R1-MSB	K1.R9-LSB	K1.R9-LSB
R1-LSB	R1-LSB	K0.R9-LSB	K0.R9-LSB
R0-MSB	R0-MSB	K1.R8-MSB	K1.R8-MSB
R0-LSB	R0-LSB	K0.R8-MSB	K0.R8-MSB
GR = 0x80808080	GR = 0x80808080	K1.R8-LSB	K1.R8-LSB
LK	LK	K0.R8-LSB	K0.R8-LSB
Type of ctxt swtch 1		K1.R7-MSB	K1.R7-MSB
		K0.R7-MSB	K0.R7-MSB
		K1.R7-LSB	K1.R7-LSB
		K0.R7-LSB	K0.R7-LSB
		K1.R6-MSB	K1.R6-MSB
		K0.R6-MSB	K0.R6-MSB
		K1.R6-LSB	K1.R6-LSB

		K0.R6-LSB	K0.R6-LSB
		K1.R5-MSB	K1.R5-MSB
		K0.R5-MSB	K0.R5-MSB
		K1.R5-LSB	K1.R5-LSB
		K0.R5-LSB	K0.R5-LSB
		K1.R4-MSB	K1.R4-MSB
		K0.R4-MSB	K0.R4-MSB
		K1.R4-LSB	K1.R4-LSB
		K0.R4-LSB	K0.R4-LSB
		K1.R3-MSB	K1.R3-MSB
		K0.R3-MSB	K0.R3-MSB
		K1.R3-LSB	K1.R3-LSB
		K0.R3-LSB	K0.R3-LSB
		K1.R2-MSB	K1.R2-MSB
		K0.R2-MSB	K0.R2-MSB
		K1.R2-LSB	K1.R2-LSB
		K0.R2-LSB	K0.R2-LSB
		K1.R1-MSB	K1.R1-MSB
		K0.R1-MSB	K0.R1-MSB
		K1.R1-LSB	K1.R1-LSB
		K0.R1-LSB	K0.R1-LSB
		K1.R0-MSB	K1.R0-MSB
		K0.R0-MSB	K0.R0-MSB
		K1.R0-LSB	K1.R0-LSB
		K0.R0-LSB	K0.R0-LSB
		K1.GR	K1.GR
		K0.GR	K0.GR
		LK	LK
		Type of ctxt switch 1	

7.4 os_cpu_a.s

A MicroC/OS-II port requires six assembly language functions. These functions are needed as you normally cannot save/restore registers from C functions. The six functions are:

```
OSCPUSaveSR()
OSCPURestoreSR()
OSStartHighRdy()
OSCtxSw()
OSIntCtxSw()
OSTickISR()
```

7.4.1 OSCPUsaveSR() and OSCPURestoreSR()

These two functions are not used in the ST100 ports. They are used to disable and enable CPU interrupts. The ST100 port uses OS_ENTER_CRITICAL #1 instead (refer to section [OS_ENTER_CRITICAL and OS_EXIT_CRITICAL](#)).

7.4.2 OSStartHighRdy()

```
// call user definable OSTaskSwHook if needed
.if KERNEL_SWITCH_HOOK > 0
```

```

        call    OSTaskSwHook
    .endif

    makea    p14, %abs16to31(OSRunning)
    make    r14, 1
    morea    p14, %abs0to15(OSRunning)
    sdb     @(p14 + 0), r14

    makea    p14, %abs16to31(OSTCBHighRdy)
    morea    p14, %abs0to15(OSTCBHighRdy)
    law     p14, @(p14 + 0)
    law SP, @(p14+0)

    .if KERNEL_SAFE_MODE > 0
        pop 0x1FFF80
        law p3, @(p15+4)
        law p2, @(p15!+8)
        poprte 0x3F
    .else
        // get the type of context switch done
        ldw r0, @(p15!+4)
        // Pop non-scratch registers only ???
        cmpeqh g0, r0, 0
        g0?goto task_switch
        // Else restore all registers
        pop 0x1FFF80
        law p3, @(p15+4)
        law p2, @(p15!+8)
        poprte 0x3F
    .endif

```

OSStartHighRdy() is called by OSStart() to start the highest priority task that was created before calling OSStart().

OSStart() sets OSTCBHighRdy to point to the OS_TCB of the highest priority task.

We call the OSTaskSwHook() function only if KERNEL_SWITCH_HOOK flag is set. This optimization is non-standard to MicroC/OS-II book.

The task stack frame is initialized as if an interrupt just occurred and all CPU registers are saved onto the interrupted task's stack. If the kernel uses the safe mode (KERNEL_SAFE_MODE flag is set), we restore all registers using *pop* and *law* instructions. If the kernel uses the default mode, we check the first value on the stack (0 for non-scratch only, 1 for all registers) and restore non-scratch (using function **task_switch**) or all registers using *pop* and *law* instructions.

To complete the return from interrupt, a *poprte* instruction is executed.

7.4.3 OSCtxSw()

This function is required only if the kernel is not compiled using the flag KERNEL_NO_TRAP. It is called by the macro OS_TASK_SW that generates a software trap #15. When KERNEL_NO_TRAP is used the macro OS_TASK_SW is a inline assembly routine.

```

(1)
    .if KERNEL_SAFE_MODE > 0
        // Push all registers
        push 0x1FFFFFFF
    .else
        // Push non-scratch registers
        push p15, p14, p4-p11, L0, L1, L2, fr, r4-r11, LK, GR

        // save on stack that it is a non-interrupt context switch
        more r0, 0
        sdw @(p15-!4), r0
    .endif

(2)
    // Save the current task's stack pointer into the current task's OS_TCB
    // OSTCBCur->OSTCBStkPtr = stack pointer

```

```

// As OSTCBStkPtr is the first field of OSTCBCur, we can use OSTCBCur
// address directly
makea p14, %abs16to31(OSTCBCur)
morea p14, %abs0to15(OSTCBCur)
law p14, @(p14 + 0)
saw @(p14+0), SP

```

(3)

```

// call user definable OSTaskSwHook if needed
.if KERNEL_SWITCH_HOOK > 0
call OSTaskSwHook
.endif

```

(4)

```

// OSTCBCur = OSTCBHighRdy
// OSPrioCur = OSPrioHighRdy
makea p1, %abs16to31(OSTCBHighRdy)
makea p2, %abs16to31(OSTCBCur)
morea p1, %abs0to15(OSTCBHighRdy)
morea p2, %abs0to15(OSTCBCur)
lduw r14, @(p1 + 0)
makea p14, %abs16to31(OSPrioHighRdy)
sdw @(p2 + 0), r14
morea p14, %abs0to15(OSPrioHighRdy)
makea p2, %abs16to31(OSPrioCur)
ldub r14, @(p14 + 0)
morea p2, %abs0to15(OSPrioCur)
sdb @(p2 + 0), r14

```

(5)

```

// stack pointer = OSPrioHighRdy->OSTCBStkPtr
// OSPrioHighRdy->OSTCBStkPtr is already stored in p1
law p1, @(p1 + 0)
law SP, @(p1+0)

```

(6)

```

// Restore all non-scratch registers from the new task's stack
// Execute a return from interrupt instruction
.if KERNEL_SAFE_MODE > 0
pop 0x1FFF80
law p3, @(p15+4)
law p2, @(p15!+8)
poprte 0x3F
.else
// get the type of context switch done
ldw r0, @(p15!+4)
cmpgth g0, r0, 0
g0?goto full_switch
// Pop non-scratch registers
poprte p15, p14, p4-p11, L0, L1, L2, fr, r4-r11, LK, GR
full_switch:
pop 0x1FFF80
law p3, @(p15+4)
law p2, @(p15!+8)
poprte 0x3F
.endif

```

1. `OSCtxSw()` save the all registers if `KERNEL_SAFE_MODE` flag is set, or only the non-scratch otherwise (and the non-scratch information on stack).
The software trap already saves the PCnext and PSR.
2. Save the current task's stack pointer into the current task's `OS_TCB`.
3. Call the `OSTaskSwHook()` function only if `KERNEL_SWITCH_HOOK` flag is set. This optimization is non-standard to MicroC/OS-II book.
4. Modify the current TCB and priority to the highest priority.
5. Change the stack pointer
6. Restore all registers if `KERNEL_SAFE_MODE` flag is set. In default mode, restore all registers (if the top of stack is 1) or only the non-scratch (if the top of stack is 0).

7.4.4 OSTickISR

(1)

```

// Push all registers
push 0x1FFFFFF

.if KERNEL_SAFE_MODE > 0
.else
// save on stack that it is an interrupt context switch
more r0, 1
sdw @(p15-!4), r0
.endif

```

(2)

```

// Set GP and hardware loops for called functions
.if KERNEL_ISR_FAST > 0
loopena
barrier
.endif
makea gp,%abs16to31(__stm_begin_gpbase)
morea gp,%abs0to15(__stm_begin_gpbase)

```

(3)

```

// Call OSIntEnter or increment OSIntNesting
// if (OSRunning == TRUE) {
// if (OSIntNesting < 255u) {
// OSIntNesting++;
// }
// }
makea p14, %abs16to31(OSRunning)
makea p4, %abs16to31(OSIntNesting)
morea p14, %abs0to15(OSRunning)
morea p4, %abs0to15(OSIntNesting)
ldub r15, @(p14 + 0)
cmpnew g1, r15, 1
ldub r15, @(p4 + 0)
g1 ? goto ._avoidAdd
cmpltuw g1, r15, 255
g1 ? add r15, r15, 1
g1 ? sdb @(p4 + 0), r15

._avoidAdd:
// Save SP into the task's OS_TCB
// if (OSIntNesting == 1) {
// OSTCBCur->OSTCBStkPtr = SP;
// }
makea p14, %abs16to31(OSTCBCur)
cmpeqw g1, r15, 1
morea p14, %abs0to15(OSTCBCur)
g1 ? law p14, @(p14 + 0)
g1 ? saw @(p14 + 0), SP

```

```

// Clear interrupting device

```

```

// Re-enable interrupts (optional)

```

(4)

```

// Call OSTimeTick
call OSTimeTick

```

(5)

```

// Call OSIntExit
call OSIntExit

```

(6)

```

// Restore processor registers
// Execute a return from interrupt instruction
.if KERNEL_SAFE_MODE > 0
pop 0x1FFF80
law p3, @(p15+4)
law p2, @(p15!+8)
poprte 0x3F
.else
// get the type of context switch done

```

```

        ldw r0, @(p15!+4)
        // Pop non-scratch registers only ???
        cmpeqh g0, r0, 0
        g0?goto task_switch
        // Else restore all registers
        pop 0x1FFF80
        law p3, @(p15+4)
        law p2, @(p15!+8)
        poprte 0x3F
    .endif

```

1. Push all registers. In default mode (no KERNEL_SAFE_MODE flag), we save all registers are saved (push 1 on stack).
2. If the kernel is compiled using -O3 option. The compiler may use hardware loops. During interruption, we have to add the loppena and barrier instructions to avoid hardware loop error. The barrier instruction is cycle consuming; this code is disabled when KERNEL_ISR_FAST is not set. It also initializes the gp register to the default value __stm_begin_gpbase. The gp register can be modified and upon interruption, it has to be set properly.
3. Increment the OSIntNesting variable. We do not call OSIntEnter() as a function call is cycle consuming.
4. Call OSTimeTick().
5. Call OSIntExit(). This function determines the highest priority task. It may call OSIntCtxSw() if a higher priority task should be scheduled.
6. Restore the processor registers.

7.4.5 OSIntCtxSw

```

(1)
    // call user definable OSTaskSwHook if needed
    .if KERNEL_SWITCH_HOOK > 0
        call OSTaskSwHook
    .endif

(2)
    // OSTCBCur = OSTCBHighRdy;
    // OSPrioCur = OSPrioHighRdy;
    makea p14, %abs16to31(OSTCBHighRdy)
    makea p12, %abs16to31(OSTCBCur)
    morea p14, %abs0to15(OSTCBHighRdy)
    morea p12, %abs0to15(OSTCBCur)
    ldw r15, @(p14 + 0)
    makea p3, %abs16to31(OSPrioHighRdy)
    sdw @(p12 + 0), r15
    morea p3, %abs0to15(OSPrioHighRdy)
    makea p12, %abs16to31(OSPrioCur)
    ldub r14, @(p3 + 0)
    morea p12, %abs0to15(OSPrioCur)
    sdb @(p12 + 0), r14

(3)
    // stack pointer = OSPrioHighRdy->OSTCBStkPtr
    // OSPrioHighRdy->OSTCBStkPtr is already stored in p1
    law p14, @(p14 + 0)
    law SP, @(p14+0)

(4)
    // Restore all registers from the new task's stack
    // Execute a return from interrupt instruction
    .if KERNEL_SAFE_MODE > 0
        pop 0x1FFF80
        law p3, @(p15+4)
        law p2, @(p15!+8)
        poprte 0x3F
    .else
        // get the type of context switch done
        ldw r0, @(p15!+4)
        // Pop non-scratch registers only ???
        cmpeqh g0, r0, 0
        g0?goto task_switch
        // Else restore all registers
        pop 0x1FFF80
    .endif

```

```

law p3, @(p15+4)
law p2, @(p15!+8)
poprte 0x3F
.endif

```

1. Call the OSTaskSwHook() function only if KERNEL_SWITCH_HOOK flag is set. This optimization is non-standard to MicroC/OS-II book.
2. Modify the current TCB and priority to the highest priority.
3. Change stack pointer
4. Restore all registers if KERNEL_SAFE_MODE flag is set. In default mode, restore all registers (if the top of stack is 1) or only the non-scratch (if the top of stack is 0).

7.4.6 Task_switch

The task_switch function is used when the kernel is compiled in default mode (no KERNEL_SAFE_MODE flag). It restores the non-scratch registers

```

task_switch:
// p15U-p14, p4-p11, LE0-LS0-LC0R-LC0, r4-r11, LK, GR
// poprte 0x13CF3D
poprte p15, p14, p4-p11, L0, L1, L2, fr, r4-r11, LK, GR

```

7.4.7 Pseudo-code for ALL ISRs

All ISRs should be written using the OSTickISR function template replacing the call to OSTimeTick() by your ISR code.

8. New Project Using MicroC/OS-II ST100 Port

8.1 Standard Project Makefile

```

#-----
#
# Copyright 2003, STMicroelectronics, Incorporated.
# All rights reserved.
#
# STMICROELECTRONICS, INCORPORATED PROPRIETARY INFORMATION:
# This software is supplied under the terms of a license agreement or
# nondisclosure agreement with STMicroelectronics and may not be copied
# or disclosed except in accordance with the terms of that agreement.
#
#-----
# System      : MicroC/OS-II
# Project Component: STlxx Kernel
# File Name   : ST122 Makefile
# History     : 2003/04/10 - Creation
#-----

(1)
# Define the path to the kernel sources
KERNEL_SRC_TREE = ../../../../

#=====
all: main

(2)
#include the kernel Makefile
include $(KERNEL_SRC_TREE)/ports/st100/st122/Makefile

CFLAGS = -tp $(DSP) -Masmkeyword -Mreentrant -g

(3)
main: main.c lib$(KERNEL).a
$(CC) -B -Mnostartup $(CFLAGS) $(INCLUDE_KERNEL) -L. -l$(KERNEL) main.c -o main my_it.s

#-----

```

```

clean: cleanKernel
    $(RM) -f main *.o *.log core lib$(KERNEL).a *~

#-----
run: all
    scst main -target=$(TARGET) -off=oce_gui

#-----

```

1. Give the path to the MicroC/OS-II installation directory.
2. Include the port ST122 or ST140 port specific Makefile
3. Add your own project files, which are linked with the kernel library.

8.2 Kernel library port specific Makefile

The port specific Makefile that creates the kernel library is located in directories:

```

ports/st100/st122/    for ST122 DSP
ports/st100/st140/    for ST140 DSP

```

It generates the kernel objects in a local directory called `os` and the kernel library `libucos_ii.a`.

The Makefile recognizes the following options:

<code>KERNEL_DEBUG</code>	compile the KERNEL in debug mode
<code>KERNEL_FAST</code>	compile the kernel using -fast mode
<code>KERNEL_SAFE_MODE</code>	save all registers during context switches
<code>KERNEL_SHOW_ASM</code>	create the assembly files
<code>KERNEL_GP16</code>	compile kernel in GP16 mode
<code>KERNEL_SMALL</code>	compile kernel in small mode
<code>KERNEL_OPTIM</code>	compile kernel to optimize performance and code size
<code>KERNEL_SWITCH_HOOK</code>	call OSTaskSwHook
<code>KERNEL_NO_TRAP</code>	do not use the trap instruction for context switch
<code>KERNEL_NO_STAS</code>	try to use the standard assembler (you may have to remove some lines in assembly code).

Using `gmake`, you can compile the kernel in debug mode using the command:

```
gmake -f Makefile KERNEL_DEBUG=1
```

9. ST100 Port Notes

9.1. Kernel library startup file `crt0.s`

The ST100 ports uses a specific `crt0.s` application startup file because the standard one provided with the compiler does not export the symbols `__start0`, `__end0`, `__start1`... which are required to initialize the tasks' stack.

The current `crt0.s` enables the traps and the interruptions. We may disable this feature when running a program using hardware interruptions. If the program crashes and reboot the target, it may crash again if the interruptions are enabled in the `crt0.s`.

9.2 Interrupt Latency

The end of the chapter 2 of MicroC/OS-II book gives the following definition:

Interrupt latency = MAX(Longest instruction, User int. disable) + Vector to ISR

Most of the kernel functions disable the interrupts to modify kernel internal variables. The following results provide the time in clock cycle of user interrupt disable for some kernel functions.

Warning: these tests results are subject to change.

For task switching test (CBS optim compilation mode):

OSTaskCreateExt	33 + 15 cycles
OSTaskCreate	34
OSTaskCreate	17
OS_TCBInit	31
OS_TCBInit	+49
OSTaskSuspend	86
OSTaskResume	90
OSTaskChangePrio	145
OS_Sched	242
OSTaskStkChk	98
OSTaskQuery	2051
OSTaskDel	112 + 62

For semaphore test (CBS optim compilation mode):

OSSemCreate	32
OSSemPend	28
OSSemPend	111
OSSemPend	42
OSSemPost	49
OSSemPost	137
OSSemAccept	17
OSSemQuery	51
OSSemDel	65

9.3 Stacks in ST122 interleaved memory

The ST122 DSP has an interleaved memory with efficient stack access. You can store the stack of each task into this area if you declare the stacks as shown below:

```
#pragma section bss = ".kernelStack"
OS_STK      TaskStartStk[TASK_STK_SIZE];
OS_STK      TaskStartStk2[TASK_STK_SIZE];
#pragma section bss = default
```

And your link file should define .kernelStack:

```
.stack ALIGN(16) PAD(2K): {} > ispace
.kernelStack : { *(.kernelStack) } > ispace
```

© Copyright 2003 STMicroelectronics.

\$Revision: 1.2 \$ \$Date: 2003-10-08 10:56:19+02 \$