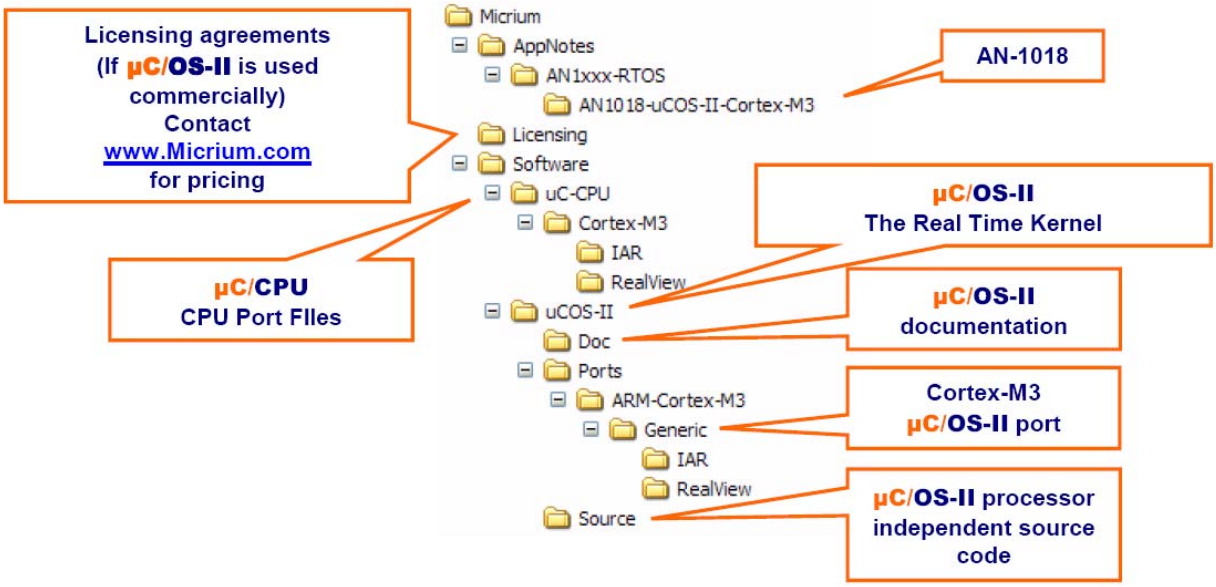
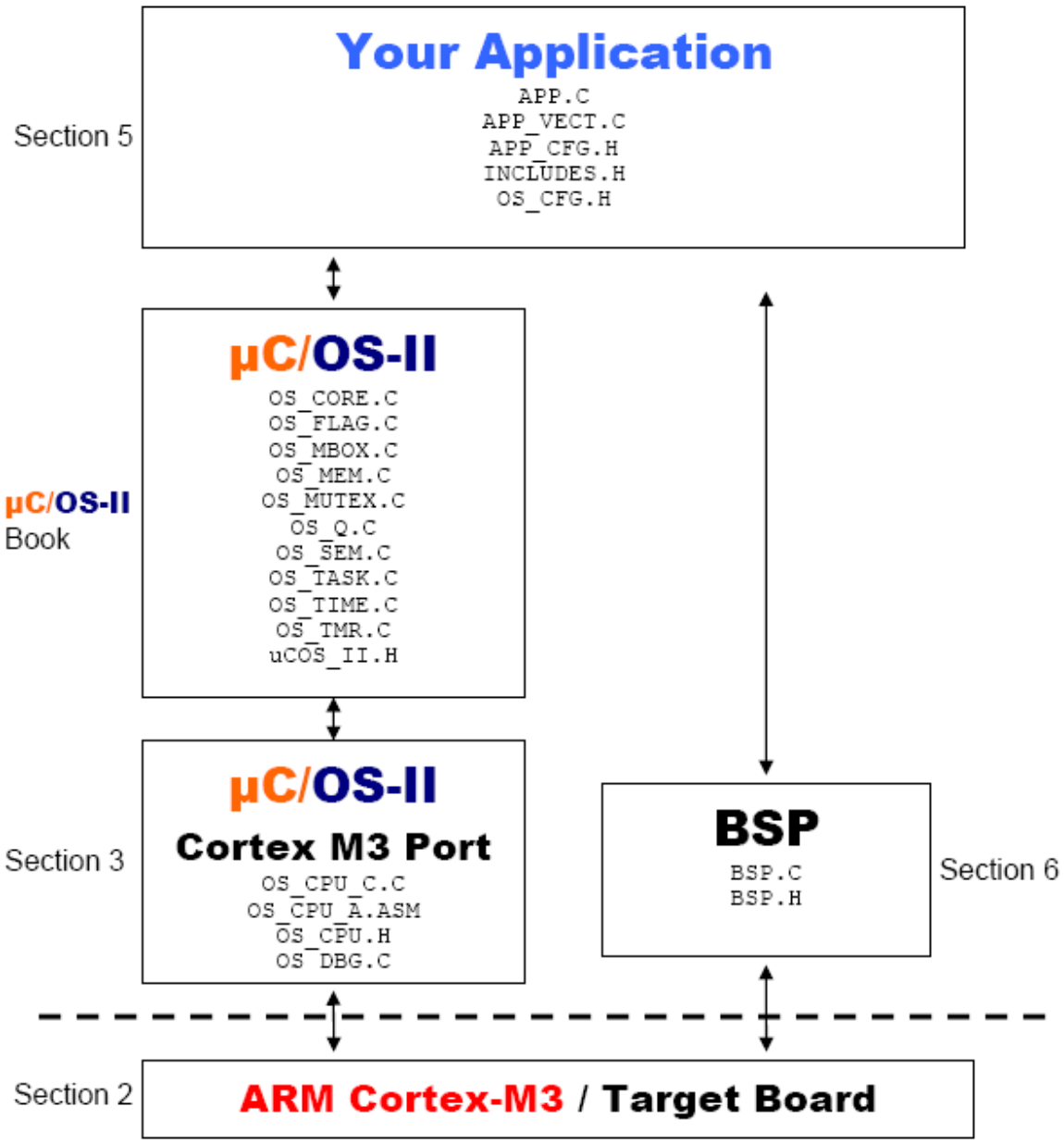
Micrium uCOS



Cool features

1) Portable



Compiler independent data types

```
typedef unsigned char  BOOLEAN;
typedef unsigned char  INT8U;
typedef signed   char  INT8S;
typedef unsigned short INT16U;
typedef signed   short INT16S;
typedef unsigned int   INT32U;
typedef signed   int   INT32S;
typedef float          FP32;
typedef double          FP64;
```

2) User runs with PSP (process stack pointer)

```
OS_CPU_PendSVHandler
    CPSID   I                   ; Prevent interruption during context switch
```

```
    MRS     R0, PSP              ; PSP is process stack pointer
    CBZ     R0, OS_CPU_PendSVHandler_nosave  ; Skip save the first time

    SUBS    R0, R0, #0x20        ; Save remaining regs r4-11 on process stack
    STM     R0, {R4-R11}

    LDR     R1, =OSTCBCur        ; OSTCBCur->OSTCBStkPtr = SP;
    LDR     R1, [R1]
    STR     R0, [R1]             ; R0 is SP of process being switched out
```

3) User can hook into OS (this is context switch)

```
    PUSH    {R14}                       ; Save LR exc_return value
    LDR     R0, =OSTaskSwHook           ; OSTaskSwHook();
    BLX     R0
    POP     {R14}
OSInitHookBegin()
OSInitHookEnd()
OSTaskCreateHook()
OSTaskDelHook()
OSTaskIdleHook()
OSTaskStatHook()
OSTaskStkInit()
OSTaskSwHook()
OSTCBInitHook()
OSTimeTickHook()
```

4) Board Support Package, Hardware Abstraction Layer, Device driver

I/O abstraction. It is often convenient to create a Board Support Package (BSP) for your target hardware. A BSP could allow you to encapsulate the following functionality:

- Timer initialization
- ISR Handlers
- LED control functions
- Reading switches
- Setting up the interrupt controller
- Setting up communication channel
- CAN, I2C, ADC, DAC, SPI, serial,graphics

```
void LED_Init(void);
void LED_On(CPU_INT08U led_id);
void LED_Off(CPU_INT08U led_id);
void LED_Toggle(CPU_INT08U led_id);
```

5) Communication and synchronization (timeout, abort)

- **Message mail box**
- **Message queue**
- **Semaphores**
- **Flags (software events)**
  - **Groups of flags**
  - **Names**
  - **pend/post, and/or**

Mutex

```
/* Description: This function waits for a mutual exclusion semaphore.
Arguments  : pevent  pointer to event control block associated with mutex.
             timeout optional timeout period (in clock ticks).
               If non-zero, your task will wait up to the specified time
               If you specify 0, however, will wait forever for resource
             perr   pointer to where an error message will be deposited.
                    OS_ERR_NONE         successful and your task owns the mutex
                    OS_ERR_TIMEOUT      not available within the 'timeout'.
                    OS_ERR_PEND_ABORT   mutex was aborted.
                    OS_ERR_EVENT_TYPE   If you didn't pass a pointer to a mutex
                    OS_ERR_PEVENT_NULL 'pevent' is a NULL pointer
                    OS_ERR_PEND_ISR     called from an ISR
                    OS_ERR_PIP_LOWER    task priority that owns is HIGHER
                    OS_ERR_PEND_LOCKED called when the scheduler is locked
* Returns    : none
* Note(s)1) The task that owns the Mutex MUST NOT pend on any other event
while it owns the mutex.
*        2) You MUST NOT change the priority of the task that owns the mutex
*/
void   OSMutexPend (OS_EVENT *pevent, INT16U timeout, INT8U *perr)
INT8U  OSMutexPost (OS_EVENT *pevent)
{
```

Other features a OS might include

1) Memory manager
2) Time delay
3) Priority resolution table
4) Debugger aware

Reference Application Note AN-1018, www.Micrium.com
MicroC/OS-II and MicroC/OS-III by Jean J Labrosse.