

smxBSP™

Board Support Package

smxBSP provides startup code and the low-level functions required for the smx® kernel and other SMX products to run on specific boards and processors. It also provides project files and other tool-support files, plus detailed BSP notes.

An smxBSP is included with smx for each board supported. (See www.smxrtos.com/eval for a complete list.) If your board is not directly supported, we provide the closest smxBSP as a porting guide. The smx kernel and other SMX RTOS products have well-defined porting layers for supporting new processors. See the psmx datasheet, smx Porting Guide, and other datasheets and manuals for more information.

BSP Code and Device Drivers

Usually we begin with startup and BSP code provided by the processor or board vendor and add any modifications needed for SMX. A UART driver and code to initialize a timer for the smx tick are included. Other SMX RTOS products (e.g. smxFS, smxNS, and smxUSB) have drivers for on-chip peripherals and also for external controller chips. The vendor often provides driver source code for other, specialized peripherals (e.g. CAN and I2C).

Tool Support

A project file is included to build the smx Protosystem for the target board. It uses the proper options and defines, and it links all needed files and SMX product libraries. It has Debug, Release, and ROM build targets, and linker/locator scripts for each case. It also includes debug configuration files necessary to pre-initialize the board before the debugger downloads code to it. In short, all files are provided that are necessary to build and run the smx Protosystem on the target board, out of the box.

BSP Notes

smxBSP includes a detailed BSP notes file for the particular target board and tool suite, which gives specific details about the BSP and configuration. It has tables showing IRQ assignments, memory maps for Debug, Release, and ROM build targets, and clock/PLL configuration. It summarizes the peripherals on the processor/board and points to the SMX drivers for each that is supported, and it has sections for each SMX product that runs on the board. It documents hardware and tool issues, and it gives instructions for using related tools such as flash programmers. We put a lot of effort into creating and updating these notes; they are an invaluable reference for us too. Each SMX Evaluation Kit includes the BSP notes for that board.

Supporting a New Board

In order to ease the job of moving SMX to a new board or processor, we have formalized the API for smxBSP. The API is listed in the next section. To support a new board or processor with SMX, it is necessary only to implement the required smxBSP functions. In many cases this consists of merely copying needed functions and making minor changes. This is especially true when moving from a vendor-supplied development board to a custom board using the same processor.

smxBSP API

The following API has evolved from supporting numerous processors in the ARM, ColdFire, and x86 processor families. It encompasses the considerable variety of ways in which interrupts,

traps, timers, DMA, memory, console, boot, and exit are handled and works quite well. See the smx Target Guide for details about each function.

In the following, *IRQ* represents a hardware interrupt, whereas *interrupt* represents either a hardware or software interrupt. IRQ configuration is done in a single array, called `irq_table[]`. Each element of the table has fields for priority, interrupt vector number, and any other details related to configuring IRQs for the processor. Centralizing this information helps prevent double-assignment of interrupt priorities and vector numbers, and it simplifies the parameter lists of some API calls. Any BSP function that needs interrupt information gets it from `irq_table[]`.

Interrupt Handling Functions

`bspClearIRQ(irq_num)`
`bspConfigureIRQ(irq_num)`
`bspEndOfIRQ(irq_num)`
`bspInitInterruptCtrl(void)`
`bspIRQToInt(irq_num)`
`bspGetIntVect(int_num, extra_info)`

`bspGetIRQVect(irq_num, xtra_info)`
`bspSetIntVect(int_num, isr_ptr)`
`bspSetIRQVect(irq_num, isr_ptr)`
`bspSetTrapVect(int_num, isr_ptr)`
`bspMaskIRQ(irq_num)`
`bspUnmaskIRQ(irq_num)`
`bspMaskIRQs(void)`
`bspUnmaskIRQs(void)`
`bspRestoreIntState(prev_state)`
`bspSaveIntStateAndDisable(void)`

Memory Functions

`bspAllocDMABuffer(num_bytes)`
`bspFreeDMABuffer(buffer)`
`bspGetHeapSpace(size, flag, heap)`

Time Functions

`bspDelayUsec(num)`
`bspInitClocks(void)`
`bspInitTick(void)`
`bspSetStime(void)`

Miscellaneous Functions

`bspExit(retcode)`
`bspInitConsoleIn(void)`
`bspInitConsoleOut(void)`
`bspInitPeripherals(void)`
`bspReboot(void)`
`bspRestart(void)`