



smxPPC[™]

Support for the PowerPC Processor Family

smxPPC is designed for the PowerPC processor family. It has many features to facilitate PowerPC development, which are discussed below. See the smx main brochure for general smx features.

Processors Supported

- 400, 500, 600, 700, 800, 5200, and 8200 Series

smxPPC uses the PIT interrupt as the time base. *smx* functions, the *smx* scheduler, and your application can be optimized for a specific processor by setting one compiler directive.

Development Tools Supported

- Metrowerks CodeWarrior Compiler/BDM Debugger with point and click *smx awareness* and the *smx* Graphical Analysis Tool (GAT)
- MetaWare High C/C++ Compiler
- Motorola Graphical Configuration Tool (GCT) for mpc5xx processors
- Diab Data C/C++ Compiler
- SDS SingleStep Simulator/BDM Debugger with point and click *smx awareness*
- Lauterbach TRACE32 with *smx awareness*

Development System Requirements

- Windows 95, 98, Millennium, NT, 2000, or XP

***smxPPC* Development Kit Contents**

- pre-made EABI-compliant ELF/DWARF *smxPPC* libraries
- Source code platform for easy start
- User's Guide, Reference Manual, and *smxPPC* Quick Start Manual
- Site development license
- Royalty-free license for one developed product

Easy Upgrade From Other Processors

smxPPC shares the same code base with *smx86*,

smxARM, and *smxCF*. Therefore it is easy to migrate between *smxPPC* and these other processor versions of *smx*. If you have experience with *smx* on one processor then you are already down the learning curve for other processors.

Stack Flexibility

Stack-pool stacks are supported by *smxPPC*. These are fixed-size stacks that are loaned to tasks, then returned to the stack pool when tasks stop, so other tasks can use them.

smxPPC also permits allocating heap stacks. Stacks may be allocated from the heap for tasks requiring stacks larger or smaller than the fixed-size stack-pool stacks.

Saving Floating Point State

smxPPC makes it easy for tasks using floating point to save and restore the floating point registers when they are suspended and resumed. Also these registers are saved only for tasks that use floating point. This way, tasks not using floating point are not burdened with the extra overhead.

Debugger Support

smxPPC supports symbolic debugging for any debugger or emulator that can read the ELF/DWARF or COFF file format.

smxAware takes symbolic debugging a step further by enabling the CodeWarrior debugger to be *smx*-aware. When the interface is enabled, *smxAware* is aware of all tasks and *smx* objects running in the system.

With *smxAware* you can:

- Set task-specific breakpoints. The breakpoint will be triggered only if it is hit while the specified

- task is running.
- Display information about kernel specific objects such as tasks, semaphores, exchanges, events, heaps, stacks etc. with the Kernel Objects dialog box under the Data menu.
- Display task specific variables in the Watch and Read windows.
- Display custom user application objects.
- Display trace strings sent from the user's code

Lauterbach TRACE32 Support

Lauterbach has implemented excellent *smx* awareness for their TRACE32 emulator consisting of:

- Display of all *smx* objects with attributes
- Analyzer listing showing task switches
- Task profiling and statistics
- Task stack usage
- Task context display
- *smx* pull-down menu

PowerPC Specifics

- 1) The PowerPC is a 32-bit flat mode processor.
- 2) The *smxPPC* scheduler is written primarily in C.
- 3) *smxPPC* was designed so applications could be programmed primarily in C. Therefore assembly macros GET_CHAR, LOCKX, PGET_CHAR, PPUT_CHAR, PUT_CHAR, STACK_CHECK and TESTX are not provided.
- 4) All initialized data is in section .data. All uninitialized data is in section .bss and code is in section .text.
- 5) Time base: The PowerPC family has a built in time base. *smxPPC* uses the Programmable Interval Timer (PIT) to generate a periodic tick interrupt. The tick (interrupt) rate should be in the range of 1 msec to 100 msec. See the *smx* User's Guide Chapter 13 for more on timing.

Boards Supported

- Motorola MPC860ADS and FADS
- Motorola MBX860
- Embedded Planet RPX-Lite 823/850/860
- Embedded Planet RPX-Classic 860/860T
- Embedded Planet RPX-Classic Low Fat
- Phast 855T
- EST SBC8260

- ESD EPPC/405GP
- IBM 405GP Walnut
- Phytec 565 Core
- Embedded Planet EP852
- Embedded Planet EP866
- EST SBC Basic 860
- ESD CPCI 405
- Espace 850
- TQ Components TQM5200
- Freescale Lite 5200

Boards using any variant of the 555/565/5200/823/850/860/405 processor are very easy to support.

Drivers for the 823/850/860/8260

- Ethernet 10baseT: SCC1, SCC2, SCC3, or SCC4¹
- Ethernet 100baseT (860T and 855T)
- PPP driver¹
- PCMCIA driver¹
- UART: SMC1, SMC2
- UART: SCC1, SCC2, SCC3, SCC4
- 823 LCD Driver²
- 823 SPI Touch Screen Driver²
- ROM to RAM load and run
- MMU support
- Data Cache
- Instruction Cache

Drivers for the 405GP

- Ethernet 100baseT¹
- PPP driver¹
- UART (FIFO mode)
- Compact Flash File driver

Drivers for the 555/565

- SCI1 and SCI2 UARTS
- CANopen

860 UART driver notes:

The SMC and SCC UARTS are designed to be flexible and easy to use. All of the UARTS can be used in character, block, or PPP mode, and all can be used simultaneously.

MMU driver notes:

smx implements an easy to use table to set up the memory and cache regions. *smx* also has a workaround for the Data Cache Corruption bug (MPC860 errata CPU6 and MPC823 errata CPU2). Because of this bug many RTOS companies do not support the data cache on MPC8xx rev A and B parts.

¹ Available with *smxNet*.

² Available with *smxPEG*.

smxPPC Function Sizes

smxfunction	cw n	cw t	dc n	dc t	smxfunction	cw n	cw t	dc n	dc t
bump_msg	360	536	392	556	nheapwalkx	580	628	584	624
bump_task	840	992	900	1032	nmallocx	480	552	484	544
clear_q	2196	2244	2248	2288	nreallocx	484	556	488	548
count	552	748	592	764	pget_char	896	1100	968	1144
count_stop	704	900	744	916	pget_char_stop	1136	1340	1200	1376
create_cx	156	264	160	248	+ pput_char	920	1124	980	1156
create_eq	136	184	144	184	pput_char_stop	1176	1380	1220	1396
create_et	144	144	152	152	put_msg	192	368	204	372
create_nbpool	160	208	216	256	qsize	580	908	656	944
create_nmsg	328	436	336	428	read_timer	192	292	216	316
create_npool	496	616	524	624	receive	984	1136	1064	1204
+ create_pool	624	1052	640	1012	+ receive_stop	900	1052	992	1124
+ create_rq	216	264	224	264	+ rel_all_block	112	192	124	204
+ create_sem	360	488	340	432	+ rel_all_msg	140	220	184	264
+ create_task	432	528	404	484	+ rel_block	68	132	68	136
create_xchg	500	720	468	616	rel_nblock	100	148	108	152
delete_cx	112	264	116	252	reset_flags	76	128	80	128
delete_eq	88	204	92	204	+ resume	516	596	528	596
delete_et	408	460	452	500	resume_to	600	676	584	672
delete_msg	120	224	120	224	sendx	680	912	716	932
delete_pool	220	372	264	400	set_flags	436	488	456	504
delete_sem	136	252	176	288	+ signalx	840	952	860	964
+ delete_task	824	928	924	1000	sleepx	332	380	396	436
deletexchg	172	324	224	356	sleep_stop	328	376	388	428
error	32	932	32	2982	+ sort_nblocks	316	340	360	380
find_next	208	312	224	320	+ startx	784	968	832	996
find_pool	140	252	132	252	start_timer	352	424	360	420
+ get_block	152	312	156	308	stop	676	756	728	796
get_msg	112	240	120	240	stop_timer	232	308	256	336
+ get_nblocks	312	384	336	400	suspend	628	708	688	756
+ go_smx	2132	2172	2480	2516	test	508	648	560	688
hook	124	204	128	204	test_flags	632	756	656	764
handle table	1144	1168	1385	1405	test_flags_stop	620	744	644	752
invoke	164	188	184	204	test_stop	796	936	860	988
keep_time	468	504	552	576	unlockx	100	176	104	180
locate	308	568	300	548	scheduler	2008	3168	3305	4693
ncallocx	76	76	80	80					
nfreex	256	304	248	288					
nheapchkx	44	44	44	44	min kernel	9648	11704	10432	12236
nheapinix	320	392	304	364	max kernel	37020	47340	40810	52496
nheapsetx	344	368	352	372					

Key: cw = CodeWarrior C/C++ v8.1, dc = Diab Data C/C++ v5.1, n = non-test mode, t = test mode
 Conditions: optimization off, event buffer logging off, DEBUGVER off
 “+” indicates calls included for a reasonable minimum kernel.

Test mode library calls contain additional code for error checking. Hence, the test mode libraries are somewhat larger (roughly 15%).

smxPPC Sample Execution Times

function	conditions	clock cycles		microsec	
		V	N	V	N
bump_task	No preempt	264	244	5.3	4.9
bump_task	Preempt	376	352	7.5	7.0
create_rq	4 levels	392	252	7.8	5.0
create_task	Unbound	184	180	3.7	3.6
create_xchg	1 level	204	184	4.1	3.7
create_xchg	2 level	268	248	5.3	5.0
interrupt latency	Maximum	30	26	0.6	0.5
interrupt response	Interrupt->isr->lsr->task	443	383	8.8	7.7
receive	Message waiting	136	124	2.7	2.5
receive_stop	Message waiting (task restarts)	404	308	8.1	6.1
send	No task waiting	242	136	4.8	2.7
send	Waiting task put into rq	292	252	5.8	5.0
start idle task	Unbound and not in a queue	356	280	7.1	5.6
stop task	In rq	288	144	5.7	2.9
task switch	Due to task suspend and resume	344	284	6.9	5.7

The number of clock cycles are valid for any PowerPC chip that can execute one instruction per clock (assuming 100% cache hit rate). All times assume a 50 MHz MPC860 with 100% cache hit rate.

V is unoptimized code with extensive stack and parameter checking. Built with the command line `mak L V 8`.

N is optimized code with basic stack checking. Built with the command line `mak L N 8`.

smxPPC RAM Usage

RAM usage is the sum of:

	<u>Sample Sizes*</u>
Stack Pool (shared stacks)	8400
Near Heap Space:	
Control Blocks	3200
lsr queue	160
Error Buffer (optional)	800
Handle Table (<i>smxDLM, smxProbe</i>)	800
Bound Stacks	a/r
Call Trace Buffer (<i>smxProbe</i>)	a/r
Task Trace Buffer (<i>smxProbe</i>)	a/r
Dynamically Allocated Regions	
Block and Message Pools	3000
<i>smx</i> Global Variables	500
Total	<u>16.5K</u>

Stack Pool: The value shown is for 7 stacks of 1200 bytes. 1200 bytes is larger than most tasks require, but it is a good starting point.

Control Blocks are generally 12 to 28 bytes. One control block is needed for every *smx* object (i.e. task, semaphore, pipe, message, etc) and for each lower level of the ready queue, semaphore, and exchanges.

Heap space: Add to these the amount of heap required by the application.

DAR's (dynamically allocated regions) are blocks of memory from which pools may be allocated.

***smx* global variables** include the configuration table and handles such as `ct` (current task) and `rq_top` (top task in ready queue).

***Sample Sizes:** These numbers reflect data usage by a medium-sized application. The stack pool usage reflects 7 medium stacks. It is better to have smaller stack pool stacks and use bound stacks where larger ones are necessary. Control block usage is a total of all control blocks needed by this particular sample application: 15 tasks, 7 stacks, 100 queue levels, 5 timers, 18 messages, 8 pipes, 2 buckets, 6 pools, and 14 blocks. The number of control blocks allocated of each type and the sizes of the heap, dar, lsr queue, handle table, error buffers are user-tuneable to the requirements of the application.