

smx86™

Support for the x86 Processor Family

smx86 supports the Intel x86 processor family, including compatibles, such as the NEC V series, AMD 18x and Elan series, National Geode, VAutomation Turbo 186, ZF Micro MachZ, and STMicroelectronics ST100. It has many features to facilitate x86 development and offers support for real-mode and 32-bit protected mode. See the smx brochure for features common to all smx versions.

80x86 Specialization

Although it now supports many other processor families, smx was originally designed solely for x86 real-mode. It was then extended to 32-bit flat protected mode (*32fpm*). More recently it was extended to support 24-bit addressing, in real-mode, for the VAutomation Turbo186 core.

Along the way, several x86 support products were added to smx86. These are summarized later in this brochure.

smx Saves Memory

smx is one of the few kernels that supports all memory models in real-mode. "This gave us a 20KB reduction in size over Nucleus" according to Carl McCauley of Checkmate Electronics. smx is the kernel for you if you are trying to shoehorn your application into limited memory. smx also has a unique stack-sharing capability that is helpful for limited-RAM applications.

smx86 Versions

Real Mode (rm)

The 16-bit real-mode (rm) version of smx86 is intended for users who are using 8086, 80186, or equivalent, processors or who do not have large applications and do not need the protection mechanisms of 386-class processors. smx86 is also used for running with DOS. (See section on DOS Support, below.)

smx86 for real mode supports the four main x86 memory models: Small, Medium, Compact, and Large. This offers optimum speed and memory

usage for small applications. An smx library is provided for each model. Regardless of the memory model, all smx variables and control blocks reside in near data memory and all smx code resides in near code memory. This gives optimum performance for smx operations.

An smx86 rm application can be built as an OMF, HEX, or BIN file using a linker/locator or built as an EXE to run under DOS or to be boot-loaded by the smx Real Mode Bootloader. (See discussion, below.) Linker/locator support includes startup code and a build script for the

Protosystem. These can be used for the application, with very little change. Locator startup code copies initialized variables from ROM to RAM. It also offers the option to start from the reset vector or to start as a BIOS extension. Paradigm C++ is supported for building located code; Borland and Microsoft 16-bit tools are supported for building an EXE.

smx86 also supports the 24-bit addressing of processors based on the Turbo186 core, known as Extended Mode. This core increases the paragraph size from 16 bytes to 256 bytes, giving a 16MB address space in real mode. For tools, Paradigm C++ must be used, along with the JTAG debug interface, VSA-186, from First Silicon Solutions.

As another example of the special support smx86 offers for the x86, it supports use of the processor 3 compiler switch, which allows the compiler to use 32-bit registers for 386-class processors. This can result in smaller and faster code if many 32-bit integer operations are done. See our white paper "Using 32-bit Registers in 16-bit Modes" in the Articles section of our web site. This option requires smx source code.

32-Bit Flat Protected Mode (32fpm)

The 32fpm version of smx86 provides 32-bit protected mode support, without segmentation. Many users prefer flat-mode addressing, because

it is simpler. This is the only 32-bit mode supported by the Microsoft and Borland 32-bit x86 compilers. In this mode, the segment registers are loaded only once, during startup. Assembly language portions of smx86 have been rewritten to take advantage of 32-bit instructions and to remove segmentation.

The services provided by smx86 for 32fpm are the same as for rm, except for the omission of the far heap, far stacks, and other far objects. A single flat-model smx library is provided. When smxSim is also purchased (see below), a special version of the smx library is provided that is suitable for execution of the Protosystem/application in a Win32 environment.

An smx86 32fpm application can be built as an OMF, HEX, or BIN file using a linker/locator or built as an EXE (PE format) to run under pmEasy32. Linker/locator support includes startup code and a build script for the Protosystem. These can be used for the application, with very little change. Startup code copies descriptor tables and initialized variables from ROM to RAM. It also offers the option to start from the reset vector or to start as a BIOS extension. Paradigm C++ is supported for building located code. Microsoft and Borland 32-bit tools and the Soft-Scope debugger are supported for building an EXE. smxSim allows EXE builds to run in a Win32 environment.

Additional x86 Features

Stack Flexibility

In addition to the stack flexibility provided by all versions of smx, segmented smx86 allows stacks to be allocated from either the near heap or the far heap on a task-by-task basis. Thus, a task requiring a near stack, so that `ss == ds`, can get its stack from the near heap. However, since the near heap is very small, this is possible only for small

stacks. For other tasks, stacks of up to almost 64KB can be allocated from the far heap. For compact and large memory models, the stack pool is in its own segment, allowing up to 64KB of stack pool space and conserving precious space in near memory. In this case, `ss != ds`, which is ok for smx (but may not be for all application code).

Coprocessor State Saving

For a task using the x87 coprocessor, smx86 will automatically save the coprocessor registers when the task is suspended and restore coprocessor registers when the task is resumed. The coprocessor's state is saved only for tasks that use it — and only in sections of the task that use it, if desired. This way, tasks not using the coprocessor are not burdened with this extra overhead (which is significant).

DOS Support

Although smx86 was designed as a ROM'able kernel, the real-mode version will run with DOS. The smx DOS Protosystem provides code to trap DOS calls so that only one task, at a time, may perform a DOS call. If a higher priority task attempts a simultaneous DOS call, it is automatically suspended on the *in_dos* semaphore until the current task completes its DOS call. The DOS

Protosystem also inhibits smx stack checking when in a DOS call because DOS switches stacks. It exits cleanly back to DOS, when the application is terminated.

Hand-tuned Scheduler

The smx86 scheduler is written in x86 assembly language and is hand-tuned for optimal performance, using many advanced x86 techniques.

PC Operation

smx86 includes a Board Support Package (BSP) for operation on standard PCs. This includes most PC-compatible boards (e.g. PC104). The smx86 BSP includes a keyboard handler, direct screen write functions, 16550 UART driver, and tick handler. Normally, the BIOS provides board initialization.

x86 Support Products

smx86 is supported by a wealth of other Micro Digital products. What follows is a brief description of each. For more information, see corresponding brochures and manuals.

smxAware for Paradigm C++ and Soft-Scope

smxAware is a DLL that is loaded by the host debugger thereby, adding smx awareness. This permits viewing smx objects and queues symbolically by selecting the smx pull-down menu. smx objects may also be placed in a watch window, and task-aware breakpoints may be set. See the smxAware datasheet for more information.

smxProbe

smxProbe permits task-level debugging. It allows the user to display control blocks, queues, and trace buffers, symbolically for all symbols entered into the smx handle table. It keeps trace information for tasks, smx calls, and smx errors. Sophisticated stop-trace points may be set. smxProbe uses the target's keyboard and display or a remote terminal connected by a serial cable. smxProbe will also operate with debuggers such as Turbo Debugger and CodeView to inspect smx objects while stopped at a breakpoint. smxProbe is included with smx86 at no additional charge.

smxWindows

smxWindows is a text-based windowing package included with smx86 that has functions for displaying and moving windows, stacking windows and changing their ordering, displaying strings, getting keyboard input and menu selections, clearing and scrolling windows, and other operations. The package requires only 10.5KB, max. It works with standard PC monitors and keyboards. Full source code is provided to permit modifying it.

Bootloader and unDOS

The Real Mode EXE Bootloader is capable of loading and executing a real-mode, EXE file from boot, without DOS present. It relies only on BIOS services. This saves the cost of DOS royalties and frees additional memory for the application. Virtually the entire RAM below 1MB is available to the application, since the space occupied by the loader becomes available, once the load has completed.

If the application makes int 21h DOS calls or certain BIOS calls, unDOS may be used to emulate those calls. This may permit a DOS application to considerably extend its memory above the 640K limit of DOS.

pmEasy and unDOS

pmEasy is used for protected mode EXE applications. It will bootload from the BIOS or can be run from the DOS command prompt. It switches the processor from real-mode to protected-mode and loads the protected-mode application from one of several possible storage devices. pmEasy also provides services to perform protected mode operations such as memory allocation, hooking interrupts, creating descriptors, etc. These services are DPMI-compatible. (The DOS Protected Mode Interface

is a standard developed by Intel, Microsoft, and others for protected-mode services.)

If loaded by DOS, pmEasy exits cleanly back to DOS when the application is terminated. This is convenient for running DOS utilities after the application performs its function.

If the application makes int 21h DOS calls or certain BIOS calls, unDOS may be used to emulate those calls. Thus, pmEasy with unDOS provides an alternative to using a DOS extender. pmEasy with unDOS is simpler than a DOS extender and it provides much better performance. This is because pmEasy was designed for real-time embedded systems, whereas DOS extenders were intended for desktop systems.

unWin

unWin includes unDOS and provides some Win32 emulation for 32fpm applications.

smxSim

smxSim permits running 32fpm smx applications under 32-bit versions of Windows. The application can be developed and debugged under Microsoft Visual Studio or the Borland Integrated Development Environment (IDE). This ability to run on the host is convenient since it avoids waiting for code to download to the target and avoids concerns about hardware startup code. It also allows getting started before the target hardware is available. When running in the IDE or directly on Windows, the application linked with smx runs as a Windows app, so it is subject to limitations imposed by Windows. smxSim, may save a few seats of expensive compiler suites. See the smxSim datasheet for more information.

smx86 Function Sizes (v3.7)

smx function	P1 S	P1 L	P3	smx function	P1 S	P1 L	P3
bump_msg	316	316	343	locate	250	250	336
bump_task	738	740	705	lsrs_on	26	26	23
clear_q	1492	1508	1521	+ misc1	834	872	794
clear_mutex	340	344	340	ncallox	92	96	43
count	434	436	486	nfreex	146	146	190
count_stop	574	576	628	nheapchkx	30	32	27
create_cx	122	126	163	+ nheapinix	216	218	269
create_dpool	782	812	823	nheapsetx	238	238	262
create_eq	94	100	115	nheapwalkx	328	328	430
create_et	104	106	99	+ nmallocx	306	308	381
+ create_fbpool	106	106	130	nreallocx	374	388	401
create_hmsgf	240	250	—	num_in_pipe	106	106	131
create_hmsgn	234	246	286	pget_char	748	750	768
create_hpoolf	414	430	—	pget_char_stop	1054	1056	1099
create_hpooln	410	428	484	pput_char	762	764	772
create_mutex	224	228	245	pput_char_stop	1070	1072	1130
+ create_rq	146	150	177	put_msg	186	190	238
+ create_sema	284	292	312	qsize	408	408	558
create_taskf	340	356	—	read_timer	198	198	178
+ create_taskn	312	336	380	+ receive	712	714	740
+ create_xchg	468	480	478	receive_stop	710	712	719
delete_cx	164	168	192	rel_all_blocks	96	96	106
delete_eq	134	140	153	rel_all_msgs	138	144	144
delete_et	320	322	323	rel_block	72	72	81
delete_hmsg	330	336	215	+ rel_fblock	104	110	112
delete_hpool	310	324	298	rel_mutex	576	578	598
delete_mutex	116	122	120	reset_flags	68	68	75
delete_sem	206	216	211	+ resume	384	384	402
delete_task	428	440	457	resume_pget	350	350	370
delete_xchg	228	238	249	resume_pput	370	370	383
dq_msg	132	132	129	+ sendx	584	590	597
+ error manager	1116	1369	1217	set_flags	376	376	362
fcallocx	202	212	—	+ signalx	536	536	625
ffreex	500	514	—	sleepx	240	248	239
fheapchkx	30	32	—	sleep_stop	240	248	236
+ fheapinix	434	440	—	sort_fblocks	262	312	292
fheapsetx	460	472	—	+ startx	590	598	615
fheapwalkx	828	844	—	start_timer	300	310	282
+ fmallocx	650	664	—	stop	454	456	467
freallocx	1134	1166	—	stop_timer	254	256	205
find_next	188	188	227	suspend	440	442	456
find_pool	122	122	143	+ test	380	382	421
free_mutex	338	342	353	test_flags	468	470	509
get_block	150	150	191	test_flags_stop	466	468	498
+ get_fblocks	342	358	362	test_stop	628	630	670
get_msg	132	132	159	unhook	92	92	91
get_mutex	614	618	616	+ unlockx	88	88	100
+ go_smx	1519	1828	1898	+ scheduler	1794	1846	1879
handle table	870	1008	841				
hook	124	148	114				
+ invoke	102	108	118	min kernel	12545	13343	12516
+ keep_time	538	566	509	max kernel	40079	41477	37114

Notes:

1. **P1 S** = real mode, small model; **P1 L** = real mode, large model; **P3** = 32-bit flat protected mode.
2. smx v3.7.1 library built with full optimizations for speed (not size). SMX_EVB = 0; SMX_PROFILING = 0. Test Mode enabled (smx error testing).

smx86 Sample Execution Times

function	conditions	processor times (microsec)			
		A	B	C	D
bump_task	No preempt	267	78	41	9
bump_task	Preempt	364	103	62	13
create_rq	4 levels	164	44	23	5
create_task		113	32	20	4
create_xchg	1 level	185	51	27	6
create_xchg	2 task and message levels	251	66	35	7
interrupt latency	Maximum	21	13	7	1
interrupt response	Interrupt->isr->lsr->task	448	120	72	15
receive	Message waiting	131	38	24	5
receive_stop	Message waiting (task restarts)	342	93	52	11
send	No task waiting	201	56	28	6
send	Waiting task put into rq	241	65	34	7
start idle task	Unbound and not in a queue	182	51	30	6
stop task	In rq	128	38	17	4
task switch	Due to task suspend and resume	268	69	58	12

processor A: 10 MHz 80C188
 processor B: 12 MHz 80286
 processor C: 20 MHz 80386
 processor D: 50 MHz 80486

All times were measured for the real-mode small model with maximum optimization.

smx86 RAM Usage

RAM usage is the sum of:	Sample Sizes*		Control Blocks are 8 to 28 bytes in size for 16-bit modes and 12 to 40 bytes for 32-bit modes. One control block is needed for each smx object (e.g. task, semaphore, pipe, message) and for each lower level of the ready queue, semaphores, and exchanges.
	16-bit	32-bit	
Stack Pool (shared stacks)	7000	9100	Heap space: Add to these the amount of heap required by the application.
Near Heap Space:			
Control Blocks	2030	3160	DAR's (dynamically allocated region) are blocks of memory from which pools are allocated.
lsr queue	80	160	
Far Heap Space:			
Error Buffer (optional)	400	600	
Handle Table	400	400	smx global variables include the configuration table and handles such as ct and rq_top.
Dynamically Allocated			
Regions			*Sample Sizes: These numbers reflect data usage by a medium-size application. The stack pool usage reflects 7 medium stacks. 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 heaps, dar, lsr queue, handle table, error buffers are user-tunable to the requirements of the application.
Block and Message Pools	2000	3000	
smx Global Variables	300	500	
Total	12K	17K	