



OpenComRTOS Layer L0/L1 code measurements for MelexCM (MLX16x8)

Release of 27-12-2006

Revision of 13-02-2007 (MelexCM Platform)

1. Static memory usage for L0 Kernel and Kernel data structures

		code size (bytes)	Ro data (bytes)	static data (bytes)	static data size (bytes) per item							
					local port	Packet	task (tcb)	task input port	task context	timer	routing table	
MP	Full	-Os	3510	24	58	6	20	22	6	6	6	2
		-O3	4004	24	58	6	20	22	6	6	6	2
	Full, No WT	-Os	2624	20	46	6	18	22	6	6		2
		-O3	2918	20	46	6	18	22	6	6		2
	Small	-Os	1756	12	34	6	16	18	6	0		2
		-O3	1960	12	34	6	16	18	6	0		2
SP	Full	-Os	2706	24	52	6	20	22	6	6	6	
		-O3	3024	24	52	6	20	22	6	6	6	
	Full, No WT	-Os	1820	20	40	6	18	22	6	6		
		-O3	1974	20	40	6	18	22	6	6		
	Small	-Os	1012	0	26	6	14	16	6	0		
		-O3	1112	0	26	6	14	16	6	0		
	Tiny	-Os	904	0	20	2	8	6	2	0		
		-O3	1038	0	20	2	8	6	2	0		

Notes:

1. MP = distributed, i.e. multi-processor (multi-node), version of OpenComRTOS. All services are transparent of the topology and processor independent.
2. SP = single-processor (single-node) version of OpenComRTOS
3. SPTiny = special derived variant of SP Small for MelexCM (MLX16x8) limited to 16 tasks (each with different priority) and 16 local ports.
4. Full = variant of OpenComRTOS supporting all services, i.e. currently:
 - L0_StartTask_W
 - L0_StopTask_W
 - L0_SuspendTask_W
 - L0_ResumeTask_W
 - L0_SendPacket_W, L0_SendPacket_NW, L0_SendPacket_WT, L0_SendPacket_A
 - L0_ReceivePacket_W, L0_ReceivePacket_NW, L0_ReceivePacket_WT, L0_ReceivePacket_A
 - L0_AllocatePacket_W, L0_AllocatePacket_NW, L0_AllocatePacket_WT
 - L0_DeallocatePacket_W
 - L0_WaitForPacket_W, L0_WaitForPacket_NW, L0_WaitForPacket_WTNote: the two phase services (L0_xxx_A, and L0_WaitForPacket_nn) are currently guaranteed to function correctly only for SP variant.
5. Full, No WT = variant of OpenComRTOS supporting all services (see Full), except:
 - L0_SendPacket_WT
 - L0_ReceivePacket_W
 - L0_AllocatePacket_WT
 - L0_WaitForPacket_WT
6. Small = variant of OpenComRTOS, supporting:
 - Supporting the following services:
 - L0_StartTask_W
 - L0_SendPacket_W
 - L0_ReceivePacket_W
 - no support for Task arguments
 - using a restricted Task context or workspace, which may limit the debugging of tasks. (only StackPointer)
 - typical use with a specific node configuration files eliminating non-essential static memory usage.
7. SPTiny = special derived variant of SP Small for MelexCM (MLX16x8) limited to 16 tasks (each with different priority) and 16 local ports.
8. Global ID size for TaskIDs and PortIDs:
 - MP Full, ML Small, SP Full: 16 bits (Local ID + Node ID, no SiteID and ClusterID)
 - SP Small, SP Tiny: 8 bits (Local ID only)
9. Static allocated stack frames **not** included in Task context above. (configurable per task)
10. Packet data size **not** included in above (compile-time configurable)
11. MP L0_StopTask_W does not yet include clean-up of waiting packets on remote nodes, only local node
12. Implementation of L0_EnterCriticalSection and L0_LeaveCriticalSection included as inline macros (mixed ASM and C).
13. Interrupt Service Routine (ISR) uses a dedicated stack.

Total static memory usage

A typical application with 2 application tasks sending and receiving packets using 2 ports.

		optimized	Kernel			application			total code size (bytes)	Total static appl data size (bytes)
			code size (bytes)	Ro data (bytes)	data size (bytes)	code size (bytes)	ro data (bytes)	data size (bytes)		
MP	Full	-Os	3510	24	58	202	0	2	3712	926
		-O3	4004	24	58	354	0	2	4358	926
	Full, No WT	-Os	2624	20	46	202	0	2	2826	840
		-O3	2918	20	46	354	0	2	3272	840
	Small	-Os	1756	12	34	202	0	2	1958	388
		-O3	1960	12	34	354	0	2	2314	388
SP	Full	-Os	2706	24	52	202	0	2	2908	652
		-O3	3024	24	52	354	0	2	3378	652
	Full, No WT	-Os	1820	20	40	202	0	2	2022	580
		-O3	1974	20	40	354	0	2	2328	580
	Small	-Os	1012	0	26	202	0	2	1214	176
		-O3	1112	0	26	354	0	2	1466	176
	Tiny	-Os	904	0	20	144	0	0	1048	98
		-O3	1038	0	20	258	0	0	1296	98

Notes:

1. MP Small, SP Small and SP Tiny used with node configuration files eliminating non-essential static memory usage.
2. Packet data size of 8 bytes.
3. MP Full and SP Full include static kernel packet pool of 10 packets. (280 bytes for above packet data size) (configurable)
4. MP Full and MP Small include static packet pool for Rx driver tasks of 5 packets. (140 bytes resp. 120 bytes for above packet data size) (configurable)
5. All variants include 1 static packet used for ISR.
6. Static allocated stack frames for each task and ISR **not** included. See *Total memory usage* below.
7. MP configured for 2 nodes.
8. Application does not include initialization of hardware (IO), timer ISR, and start of OpenComRTOS.
9. Application does not include C library functions like start-up and memcpy.
10. Application tasks use only L0_SendReceivePacket_W and L0_ReceivePacket_W kernel services.

2. Total memory usage

A typical application with 2 application tasks sending and receiving packets using 2 ports.

		optimized	total code size (bytes)	total static appl data size (bytes)	max used stack size (bytes)					total ram size used (bytes)
					kernel	idle	ISR	task1	task2	
MP	Full	-Os	3712	926						
		-O3	4358	926						
	Full, No WT	-Os	2826	840						
		-O3	3272	840						
	Small	-Os	1958	388						
		-O3	2314	388						
SP	Full	-Os	2908	652						772
		-O3	3378	652						776
	Full, No WT	-Os	2022	580	34	2	16	42	42	700
		-O3	2328	580	34	2	16	44	44	704
	Small	-Os	1214	176	26	2	16	36	36	276
		-O3	1466	176	32	2	16	40	40	290
	Tiny	-Os	1048	98	26	2	16	36	36	198
		-O3	1296	98	30	2	16	38	38	206

Notes:

1. Measurements on MLX81001A TBC evaluation board configured without LIN.
2. MP Small, SP Small and SP Tiny used with node configuration files eliminating non-essential static memory usage.
3. Packet data size of 8 bytes; 1 byte of data transferred in each send/receive paired transaction.
4. MP Full and SP Full include static kernel packet pool of 10 packets. (280 bytes for above packet data size) (configurable)
5. MP Full and MP Small include static packet pool for Rx driver tasks of 5 packets. (140 bytes resp. 120 bytes for above packet data size) (configurable)
6. All variants include 1 static packet used for ISR.
7. Static allocated stack sizes: 128/32/32/128 bytes for respectively kernel task, idle task, ISR, and each application task.
8. MP configured for 2 nodes.
9. Application does not include initialization of hardware(IO), timer ISR, and start of OpenComRTOS.
10. Application does not include C library functions like start-up and memcpy.
11. Application tasks use only L0_SendReceivePacket_W and L0_ReceivePacket_W kernel services.

3. Performance and interrupt latency

A typical application with 2 application tasks sending and receiving packets using 2 ports.

		performance (#packets / second)			interrupt latency (microseconds)			
		optimized	send- receive pair round- trip	receive packet (always packet available)	to first access in ISR	till injectPacket (end of ISR)	till first access in appl task waiting	Back- ground task interruption time
SP	Full	-Os						
		-O3						
	Full, No WT	-Os	7510	13888	4	13	68	117
		-O3	7680	14285	4	13	65	114
	Small	-Os	8710	16129	4	13	60	104
		-O3	9010	16666	4	13	57	100
	Tiny	-Os	9800	17543	4	11	56	98
		-O3	10740	19230	4	9	52	89

Notes:

1. Measurements on MLX81001A TBC evaluation board running 27MHz configured without LIN. (average microcontroller performance: 4 instructions/cycle or about 6.5 Mips)
2. “# receive packets” figure has been calculated from measurement of completion a single L0_ReceivePacket_W service, and which are estimated with a 10-15% accuracy.
3. MP Small, SP Small and SP Tiny used with node configuration files eliminating non-essential static memory usage.
4. Packet data size of 8 bytes; 1 byte of data transferred in each send/receive paired transaction.
5. Static allocated stack sizes: 128/32/32/128 bytes for respectively kernel task, idle task, ISR, and each application task.
6. MP configured for 2 nodes.
7. Application tasks use only L0_SendReceivePacket_W and L0_ReceivePacket_W kernel services.

4. Conclusions

The code has been fully written in C, except for the context switch and L0_enterCriticalSection and L0_leaveCriticalSection functions.

The SP Tiny variant for MelexCM uses the special MLX16x8 instructions "fsb" and "sfb" for manipulation of the packet waiting lists and task ready list.

An Excel sheet to estimate the memory usage for an application for various OpenComRTOS variants on a single processor, i.e. node, with the following parameters:

- # application tasks assigned to the node
- # local ports assigned to the node
- Packet data size (in bytes)
- Size of the kernel packet pool (in #packets) (only used for MP Full and SP Full)
- #calls to L0_SendPacket_W and L0_ReceivePacket_W per application task (other services not supported in this sheet)
- #bytes communicated per call to L0_SendPacket_W and L0_ReceivePacket_W
- #interrupt service routines

For the MP variants:

- # Nodes in the system (only used for routing table)
- # Rx driver tasks on the node
- # Tx driver tasks on the node
- Size of the Rx packet pool (in #packets)



C:\temp\
opencomrtosdata_cal

Note: the static memory usage figures for the L0 Kernel and the application dependent kernel data structure are calculated with fairly large confidence and precision; the dynamic memory usage figures are highly dependent on the application.

5. Relative performance measurements on a Windows NT platform

Platform: DELL Inspiron 6000 with 1.60 GHz Intel Pentium (single core) and 720 MBytes of RAM.

Platform	variant	#iterations	time(s)	datasize in packet (bytes)	compiler	
WIN32	SP	100,000	3.7	16	gcc -Os	no trace, no debug
WIN32	SP	100,000	3.4	0	gcc -Os	no trace, no debug
WIN32	SPL1	100,000	3.7	16	gcc -Os	no trace, no debug
WIN32	SPL1	100,000	3.4	0	gcc -Os	no trace, no debug
WIN32	SPL1 SMALL	100,000	3.2	0	gcc -Os	no trace, no debug
WIN32	MP2 L1 P1 N1 P2 N1	10,000	11.8	0	gcc -Os	no trace, no debug idle task = Sleep(0)
WIN32	MP2 L1 P1 N1 P2 N2	10,000	11.6	0	gcc -Os	no trace, no debug idle task = Sleep(0)
WIN32	MP2 L1 P1 N1 P2 N1	10,000	12.9	16	gcc -Os	no trace, no debug idle task = Sleep(0)

SP: all entities on a single simulated node. All kernel interacting using the native Windows threading mechanism.

MP: entities are on multiple simulated nodes. All node interaction is simulated through native socket libraries.

MP2 means: MP simulation with 2 nodes

P1 N1

P2 N2

means: 2 ports, one on each node

P1 N1

P2 N1

means: 2 ports, both on same node (node1)

Note that the values for SP come without much spread. (0.1s). For MP, the spread is larger (0.5s)

Conclusion:

These test essentially measure the overhead of the Windows multi-threading and socket libraries. The results confirm that Windows XP is not suitable for real-time processing and that the overhead for context switching is very high. When compared with the "native" SP implementation on the 6.5 Mips MLX16, we obtain a Windows equivalent performance of about 15 Mips.

6. Annexes

Specifications for performance and interrupt latency measurements.

a) performance (#send/receive packets / second) and dynamic stack usage for each task for 2 task application with:

Task1 : do { sendpacket PortA; receivepacket PortB } while (1)

Task2: do { receivePacket PortA; sendpacket PortB } while (1)

i.e. the original 2 task send/receive example

b) interrupt latency:

time from interrupt to first statement, e.g. access to data, in ISR

time from interrupt till injectPacket

time from interrupt till use in application Task1 waiting for such a packet

task interruption time of background Task2

for 1 ISR and 2 task application with:

ISR: periodic timer interrupt : Send ISRPacket PortA

Task1: do { receivePacket PortA ; } while (1)

Task2: do { } while (1)

i.e. your first ISR application, except with empty Task2, or optionally your Task2 do { sendPacket PortA } while (1)

c) performance (#receive packets) and interrupt latency:

time from interrupt to first statement, e.g. access to data, in ISR (should be same as for b))

time from interrupt till injectPacket (should be same as for b))

time from interrupt till use in application Task1 waiting for PortB (should be same as for b)))

time from start of ReceivePacket Port A till return of ReceivePacket PortA in Task 1

task interruption time of background Task2

for 1 ISR and 2 task application with:

ISR: periodic timer interrupt which does alternately i) Send ISRPacket1 to PortA, ii) Send ISRPacket2 to Port B

Task1: do { receivePacket PortB ; receivePacket Port A} while (1)

Task2: do { } while (1)

d) performance (#send packets) and interrupt latency:

Same as c) but with ReceivePacket and SendPacket reversed. I.e. ISR should do a ReceivePacket PortB/PortA , and Task1 a SendPacket PortB/PortA .

7. Preliminary L1 codesize figures

OpenComRTOS L1 code size figures (MLX16)				
	MP FULL		SP SMALL	
	L0	L1	L0	L1
L0 Port	162		132	
L1 Hub shared		574		400
L1 Port		4		4
L1 Event		68		70
L1 Semaphore		54		54
L1 Resource		104		104
L1 FIFO		232		232
L1 Resource List		184		184
Total L1 services		1220		1048
Grand Total	3150	4532	996	2104

8. Document history

Date	Version	Author	Change
27.01.2006	1.0	Gjalt de Jong	Document creation Static memory OpenComRTOS L0 27.01.2006 (cf. OpenComRTOS Layer 0 Architecture and Design Document rev
14.08.2006	1.1	Gjalt de Jong	Static and dynamic memory OpenComRTOS L0 09.08.2006 MelexCM. (cf. OpenComRTOS Layer 0 Architecture and Design Document rev 1.0.13) Interrupt latency
17.08.2006	1.2	Gjalt de Jong	Added application data Added calculation sheet
13.09.2006	1.3	Andrey Farafonov	Updated profiling data for OpenComRTOS 11.09.2006 MelexCM
16.02.2007	1.4	Gjalt de Jong	Updated memory figures for OpenComRTOS 13.02.2007 MelexCM
23.04.2007	1.5	Eric Verhulst	Updated with Win32 figures Updated wit L1 data