Application Note AN076

Konsistente Zugriffe auf Parameter- und Datenbereiche von Tasks durch eine neue Semaphoren-Steuerung und ohne Nutzung der R- und W-Pointer einer Task

Autor: hb Datei: AN076.doc (4 Seiten)

Um die im Titel angesprochene Konsistenz zu erreichen, wurde bisher die Möglichkeit genutzt, die Interrupt-fähigkeit der CPU vorübergehend zu unterbinden. Im folgenden werden nun Erweiterungen des Betriebssystems OsX und der zugehörigen Bibliotheken von MODULAR-4/486 und Multi-COM ab ML8-3B.11x, ML7-3B.11x bzw. ML6-1A.09x beschrieben. Ab diesen Versionen des Betriebssystems sind Erweiterungen und Ergänzungen im Betriebssystem, in allen onboard und in den PC-Bibliotheken vorhanden.

Um den Datenbereich und den Parameterbereich einer Task mit je einer Semaphore zu schützen, gibt es vier neue Funktionen: GET_DATA_SEMA, RESET_DATA_SEMA, GET_PAR_SEMA und RESET_PAR_SEMA.

Zugriff auf Parameter

Um auf einzelne Parameter (Bit, Byte, Wort, Doppelwort) einer Task konsistent zugreifen zu können, können die bestehenden Funktionen READ_PAR_BYTE, READ_PAR_WORD, READ_PAR_DWORD, WRITE_PAR_BYTE, WRITE_PAR_WORD und WRITE_PAR_-DWORD eingesetzt werden. Die Funktionen READ_PAR_BLOCK und WRITE_PAR_BLOCK dagegen sind durch Interrupts unterbrechbar, und dann ist die Konsistenz nicht mehr garantiert. Hier konnte bisher nur DISABLE_INTERRUPT helfen, weil es keine Semaphoren gab bzw. diese nicht genutzt wurden.

Eine andere Möglichkeit des Zugriffs auf Parameter besteht darin, sich zunächst einmal die Adresse des Parameterbereichs oder eines bestimmten Parameters zu holen (mit der Funktion GET_PAR_ADDRESS) und dann direkt auf die Adresse des Parameters zuzugreifen. Die Adressen aller Parameter im Paramterbereich einer Task in OsX sind statisch. Man muß die Adresse also nur einmal ermitteln, und zwar entweder mit der Funktion GET_PAR_ADDRESS oder durch Lesen der Anfangsadresse der TDT (mit GET_TASK_INFO) und addieren der Länge der TDT (kann ebenfalls mit GET_TASK_INFO ermittelt werden). Die Problematik der Konsistenz bleibt aber unverändert erhalten, die Lösungen sehen genauso aus wie gerade beschrieben.

Abhilfe schaffen hier jetzt zwei neue Funktionen: GET_PAR_SEMA und RESET_PAR_SEMA. Mit GET_PAR_SEMA kann sich eine Task eine Semaphore eines Parameterbereichs einer anderen oder auch der eigenen Task holen, mit RESET_PAR_SEMA gibt sie sie wieder frei. Natürlich müssen logischerweise alle Tasks, die auf den Parameterbereich derselben Task zugreifen

wollen, immer zunächst versuchen, sich die zugehörige Semaphore zu holen. Die Task, die die Semaphore bekommen hat, kann dann mit allen angegebenen Funktionen konsistent auf den Parameterbereich zugreifen. Danach muß sie sie wieder freigeben. Die einzelnen Zugriffe selbst kümmern sich um die Semaphore nicht, gleichgültig, ob auf einzelne Parameter, auf einen Block oder ob direkt über die Adresse von Parametern zugegriffen wird.

Zugriff auf den Datenbereich einer Task

Für den Zugriff auf den Datenbereich einer Task besteht dasselbe Problem des konsistenten Zugriffs. Auch hier wurde durch zwei neue Funktionen GET_DATA_SEMA und RESET_DATA_SEMA eine einfache Lösung geschaffen.

Zugriff auf den Datenbereich einer Task ohne Nutzung der R- und W-Pointer der Task

Bei einigen Anwendungen kann das Problem darin bestehen, daß es nur je einen R- und W-Pointer je Task gibt. Wenn zwei Tasks auf denselben Datenbereich z.B. lesend zugreifen wollen, können sie dafür nicht denselben Pointer nutzen, weil der R-Pointer von der anderen Task verändert wurde. Deshalb ist es sinnvoll, daß jede Task, die auf einen Datenbereich zugreifen will, den Pointer dafür selber verwaltet. Hierfür stehen jetzt die neuen Block-Transfer-Funktionen READ_DATA und WRITE_DATA zur Verfügung, die nach jedem Zugriff den Pointer zwar verändern, ihn aber selbst nicht nutzen. Bei jedem Aufruf wird der Offset als 32-Bit Wert zum Anfang des Datenbereichs der Task mit übergeben. Dabei muß allerdings beachtet werden, daß alle anderen Funktionen, die für den Zugriff auf den Datenbereich zur Verfügung stehen, nicht auf dieselbe Task angewendet werden dürfen, also RESET_R_POINTER, MOVE_R_POINTER, READ_DATA_BYTE, READ_DATA_WORD, READ_DATA_DWORD, READ_DATA_BLOCK, RESET_W_POINTER, MOVE_W_POINTER, WRITE_DATA_BYTE, WRITE_DATA_WORD, WRITE_DATA_DWORD und WRITE_DATA_BLOCK, gleichgültig von welcher Task oder CPU aus.

Versionen von Treibern und Bibliotheken mit den neuen Funktionen

Karte	Typ	OsX, Treiber, Bibliothek	Version
MODULAR-4/486	ML8	OsX	ML8-3B.11x
	ML7	OsX	ML7-3B.11x
Multi-COM	ML6	OsX	ML6-1A.09x

x = R := RAM-Version, x = E := EPROM-/Flash-Version

Neue System-Routinen (Asscmbler)

GET_PAR_SEMA **(Nr. 112)**

Diese Funktion versucht, die Semaphore für den Parameterbereich der Task t zu holen.

Entry:	ax	Task-Nr. t des Parameterbereichs
Changed:	f, eax	
Exit (CY=0):	ax	Ergebnis: 0 = ok, Semaphore bekommen
		<> 0 = Semaphore nicht bekommen
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

RESET_PAR_SEMA **(Nr. 113)**

Diese Funktion gibt die Semaphore für den Parameterbereich der Task t wieder frei.

Entry:	ax	Task-Nr. t des Parameterbereichs
Changed:	f, eax	
Exit (CY=0):	-	kein Fehler
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

GET_DATA_SEMA **(Nr. 114)**

Diese Funktion versucht, die Semaphore für den Datenbereich der Task t zu holen.

Entry:	ax	Task-Nr. t des Datenbereichs
Changed:	f, eax	
Exit (CY=0):	ax	Ergebnis: 0 = ok, Semaphore bekommen
		<> 0 = Semaphore nicht bekommen
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

RESET_DATA_SEMA **(Nr. 115)**

Diese Funktion gibt die Semaphore für den Datenbereich der Task t wieder frei.

Entry:	ax	Task-Nr. t des Datenbereichs
Changed:	f, eax	
Exit (CY=0):	-	kein Fehler
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

READ_DATA_OFFS **(Nr. 116)**

Diese Funktion liest einen Datenblock aus dem Datenbereich einer Task ohne Verwendung des R-Pointers der angesprochenen Task.

Entry:	dx	Task-Nr. des Datenbereichs, der gelesen werden soll
	ebx	Offset zum Anfang des Datenbereichs der Task
	eax	Ziel-Adresse, wo die gelesenen Daten hin sollen (physikal.)
	cx	Anzahl Byte zu lesen
Changed:	f, eax, ebx, cx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_OFFS **(Nr. 117)**

Diese Funktion schreibt einen Datenblock in den Datenbereich einer Task ohne Verwendung des W-Pointers der angesprochenen Task.

Entry:	dx	Task-Nr. des Datenbereichs, in den geschrieben werden soll
	ebx	Offset zum Anfang des Datenbereichs der Task
	eax	Quell-Adresse, wo die zu schreibenden Daten stehen (physikal.)
	cx	Anzahl Byte zu schreiben
Changed:	f, eax, ebx, cx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode