SIEMENS

SICOMP IMC

RMOS3-PCI V1.0
Betriebs-Software

Technische Beschreibung

Inhalt

Vorläufige Ausgabe **25.02.02 (R3)**

Für Vollständigkeit und Richtigkeit aller Angaben wird keine Gewähr übernommen.

Februar 2002

Sicherheitstechnische Hinweise

Diese Dokumentation enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise sind durch ein Warndreieck hervorgehoben und je nach Gefährdungsgrad folgendermaßen dargestellt:

Gefahr

bedeutet, daß Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten **werden**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Warnung

bedeutet, daß Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten **können**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Vorsicht

bedeutet, daß eine **leichte** Körperverletzung oder ein Sachschaden eintreten können, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Hinweis

ist eine wichtige Information über das Produkt, die Handhabung des Produktes oder den jeweiligen Teil der Dokumentation, auf den besonders aufmerksam gemacht werden soll.

Bestimmungsgemäßer Gebrauch

Warnung

Das beschriebene Produkt darf nur für die im Katalog und in dieser Dokumentation vorgesehenen Einsatzfälle und nur in Verbindung mit von Siemens empfohlenen bzw. zugelassenen Fremdgeräten und -komponenten verwendet werden.

Inhalt

Tabellen

Einführung 1

Am PCI-Bus sind nur 4 Interrupts INTA#, INTB#, INTC# und INTD# definiert. Um bei dieser beschränkten Anzahl trotzdem mehr als 4 Interrupts im System zu ermöglichen, dürfen sie geshared werden, d.h., mehrere Interruptquellen können denselben Interrupt benutzen.

Das hat im wesentlichen zwei Konsequenzen:

- Die Interrupts sind pegelgetriggert definiert, im Gegensatz z.B. zum SMP16-Bus, dessen nicht sharebare Interrupts flankengetriggert definiert sind.
- Die Interruptquellen müssen einen Mechanismus zur Verfügung stellen, mit dem die Interrupt-Service-Routinen erkennen können, ob ein aktiver Interrupt von dem der Routine zugeordneten Interruptquelle verursacht worden ist.

Außerdem ist die Zuordnung einer Interruptquelle zu einer Interruptleitung (INTA#, INTB#, INTC#, oder INTD#) und weiter zu einer Interruptnummer nicht fest durch die Baugruppen- und Rückwandkonfiguration vorgegeben. Statt dessen wird diese Zuordnung vom BIOS während des Systemstarts erstellt und die von PCI-Interrupts genutzten Eingänge der Interruptcontroller auf Pegeltriggerung umgestellt. Die einer Baugruppe zugeordnete Interruptnummer wird im Configuration-Space dieser Baugruppe hinterlegt und kann ausgelesen werden.

RMOS3 V3.20 unterstützt nur ungesharte Interrupts, d.h. einer Interruptnummer kann nur eine einzige Interrupt-Service-Routine zugewiesen werden. Außerdem wird von flankengetriggerten Interrupts ausgegangen, d.h. eine zweite Aktivierung desselben Interrupts wird nur dann erkannt, wenn vorher das Interruptsignal zwischenzeitlich deaktiviert worden war.

RMOS3-PCI erweitert RMOS3 V3.20 um die Fähigkeit sharebare, pegelgetriggerte Interrupts vom PCI-Bus zu verarbeiten.

1.1 Produktstruktur

Das Softwarepaket RMOS3-PCI enthält folgende Komponenten für RMOS3 V3.20:

- Interrupt Server: RMISHSRV.LIB und RMISHSRV.DRV
- Interrupt Client: RMISHCLI.LIB mit den Funktionen:
    - RmInitShIntClient
    - RmSetShIntISHandler1
    - RmSetShIntISHandler2
    - RmClrShIntISHandler
- PCI-Scanner: RMPCISCA.LIB mit der Funktion:
    - RmPciSearchFunction
    - RmPciSearchSubFunction
- Headerfile rmpci.h
- Zusatzprogramme
    - LOADER (Programm zum Laden des nachladbaren RMISHSRV.LIB)
- Beispiele

Die PCI-Server-Funktionen sind sowohl in statischer Form (als Libraries) als auch als nachladbare Programme realisiert.

In der nachladbaren Version kann der RMISHSRV-Treiber mit dem Ladeprogramm LOADER.386 geladen werden.

1.2 Systemvoraussetzungen

Auf dem Entwicklungsrechner muß RMOS3 V3.20 installiert sein. Die dafür nötigen Hardwarevoraussetzungen können Sie Ihrer RMOS-Dokumentation entnehmen.

Hinweis

Für eine Systemgenerierung anhand dieser Beschreibung sind grundlegende RMOS-Kenntnisse erforderlich. Entsprechende Schulungen werden von Siemens angeboten und sind bei Ihrer zuständigen Geschäftsstelle zu erfragen!

1.3 Lieferumfang

Die Produktdiskette des RMOS3-PCI enthält folgende Komponenten:

- Shared Interrupt Server ladbar: RMISHSRV.DRV
- Shared Interrupt Server statisch: RMISHSRV.LIB
- allgemeiner Lader
- Shared Interrupt Client: RMISHCLI.LIB
- PCI-Scan RMPCISCA.LIB
- Headerfile rmpci.h
- Applikationsbeispiel

Inbetriebnahme 2

2.1 Installation

Um das Softwarepaket RMOS3-PCI zu installieren, gehen Sie wie folgt vor:

1. Falls noch nicht geschehen, installieren Sie RMOS3 V3.20 auf Ihrem Entwicklungsrechner.
2. Legen Sie die Produktdiskette von RMOS3-PCI ein.
3. Starten Sie das Installationsprogramm INSTALL.EXE auf der Diskette.
4. Folgen Sie den Anweisungen des Programms am Bildschirm.

Die installierten Verzeichnisse und Dateien sind in der ASCII-Datei READ_PCI.1P0 aufgelistet.

2.2 Konfiguration des Shared Interrupt Servers

Der Shared Interrupt Server kann sowohl statisch zum System gebunden werden, als auch zur Laufzeit geladen werden. Abhängig von der gewünschten Betriebsart sind unterschiedliche Systemkonfigurationen zu erstellen. Im Folgenden werden die notwendigen Ergänzungen zu den Standard-Systemkonfigurationen von RMOS3 V3.20 erläutert.

Der Server darf nur einmal im System geladen werden.

2.2.1 Generieren eines statischen Systems

In einem statischen System wird der RMOS3-PCI Server (RMISHSRV.LIB) mit dem System gebunden und bereits im Anlauf initialisiert.

GENSYSC.BAT:

```
REM --- PCI Interupt Sharing
ECHO ...\RMISHSRV.LIB                          >>BND_APL.CMD
```

Beispiel RMCONF.C :

```
/* RMOS3 PCI interrupt sharing */
extern int _FIXED _FAR RmIntShSrv(void);



printf("\n(c) Siemens AG, RMOS3-PCI Initialization Vx.y\n");
RmRetVal = RmIntShr();

if (RmRetVal == RM_IS_ALREADY_CATALOGED
   printf(„RMOS3CI Error: Server is already catalogud \n“);

else if (RmRetVal == RM_IS_NOT_CATALOGED)
   printf(„RMOS3PCI Error: Could not catalogued server. \n“);

else if (RmRetVal != RM_OK)
   printf(„RMOS3PCI Error: init server \n“);

else if (RmRetVal == RM_OK)
   printf(„RMOS3PCI: init server   ok \n“);
```

2.2.2 Generieren eines nachladbaren Systems

In einem nachladbaren System wird das RMOS3-PCI Server nach dem Systemhochlauf geladen

Vom CLI oder in der RMOS.INI kann der RMOS3-PCI Server mit Hilfe des Ladeprogramms LOADER.386 geladen werden. Der PCI-Treiber liegt hierfür als ladbare Version vor (RMISHSRV.DRV).

Beispiel RMOS.INI

```
run=A:\BIN\LOADER.386 A:\BIN\RMISHSRV.DRV
```

2.2.3 Meldungen

Nach dem Laden des Servers erscheint folgende Ausgabe:

Anzeige auf Konsole	Bedeutung
© Siemens AG, RMOS3PCI Initialization Vx.y RMOS3PCI: init server ok.	Erfolgreiche Initialisierung
RMOS3PCI Error : Server is already catalogued.	*RMOS3PCI* schon im Katalog gefunden.
RMOS3PCI Error : Could not catalogue server.	*RMOS3PCI* konnte nicht im Katalog eingetragen werden.
RMOS3PCI Error : init server	konnte Treiber nicht initialisieren

2.3 Ladeprogramm zum Nachladen von RMOS3-PCI

Um den RMOS3-PCI Server nachladen zu können, wird das Zusatzprogramm LOADER.386 benötigt.

Dieses Programm lädt den RMOS3-PCI-Treiber in den Hintergrund und beendet sich danach selbst. Es kann wahlweise manuell aus dem CLI oder automatisch durch das RUN-Kommando aus RMOS.INI aufgerufen werden.

Aufruf

```
LOADER.386 <path\taskname.ext> [arg2] [arg3]...
```

Aufrufparameter

Parametername	Bedeutung
<path>	Kompletter Pfad des Verzeichnisses, in dem sich das zu ladende Programm befindet. z.B. A:\BIN\
<taskname>	Name des zu ladenden Programms. z.B. RmIShSrv
<ext>	Erweiterung des zu ladenden Programms. z.B. DRV
<arg n>	Übergabeparameter des zu ladenden Programms.

Wenn das Ladeprogramm ohne Aufrufparameter gestartet wird, zeigt es auf dem Bildschirm seine Aufrufsyntax an.

Beispiel für Aufrufe aus dem CLI

```
A:\BIN\LOADER.386 A:\RMOS3PCI\RMISHSRV.DRV
```

Beispiel für Aufrufe aus RMOS.INI

```
run=C:\BIN\LOADER.386 C:\RMOS3PCI\RMISHSRV.DRV
```

Fehlermeldungen

```
LOADER_ERROR: FILE NOT FOUND
```

Die angegebene Datei ist (im angegebenen Pfad) nicht vorhanden.

```
LOADER_ERROR: ALLOC_ERROR
```

Das Ladeprogramm konnte seinen benötigten Speicher nicht ordnungsgemäß zuweisen.

2.4 Systemkomponenten

Bibliotheksdateien

Tabelle 2–1 Benötigte Bibliotheksdateien

Dateiname	Bedeutung	1)
RMISHSRV.LIB	Shared Interrupt Server	S
RMISHCLI.LIB	Shared Interrupt Client	I
RMPCISCA.LIB	PCI-Bus Scan	I

1) Verwendung
S Zur Systemgenerierung mit Cadul-Tools
I Zur Generierung einer Anwenderapplikation mit Cadul-Tools

Header-Datei

Tabelle 2–2 Header-Dateien für Socket-Anwendungen

Dateiname	Bedeutung
RMPCI.H	Funktionsprototypen.

Konfigurations-Datei

Tabelle 2–3 Konfigurationsprogramme

Dateiname	Bedeutung
LOADER.386	Ladeprogramm für RMOS3-PCI

RMOS3-PCI API 3

Der Treiber RMOS3-PCI besteht aus folgenden 3 Teile:

Der Shared Interrupt Server beinhaltet alle Funktionen um Interrupts zu sharen und zu verwalten.

Der Shared Interrupt Client dient als Schnittstelle zwischen Anwenderapplikation und Shared Interrupt Server. Der Client stellt die Möglichkeit zu Verfügung Interrupt-Service-Routinen in den Server einzuhängen und zu löschen.

Der PCI-Scanner ermöglicht das Finden bestimmter Baugruppen, oder aller Baugruppen am PCI-Bus.

3.1 Scan des PCI-Buses

Die Funktionen des PCI-Scanners RMPCISCA.LIB ermöglichen einer RMOS-Task alle PCI-Devices am Bus zu suchen oder gezielt nach PCI-Devices zu suchen, deren Vendor-ID und Device-ID, oder deren Vendor-ID, Device-ID, Subsystem-VendorID und Subsystem ID bekannt sind (siehe Technische Beschreibung der jeweiligen Baugruppe).

Die Configuration-Spaces dieser Devices bzw. dieser Functions (für den Fall von Mulit-Function-Devices) werden ausgelesen und in der Struktur PCI_CONFIG abgelegt. Die Reihenfolge des Eintrages erfolgt PCI-konform.

Der Interrupt-Client benötigt aus der PCI_CONFIG-Struktur die den Devices zugeordneten Interruptnummern.

3.1.1 RmPciSearchFunction

Funktion **Scant den PCI-Bus (in RMPCISCA.LIB enthalten)**

Syntax

```
#include <rmpci.h>
int RmPciSearchFunction ( ushort VendorID,
                          ushort DeviceID,
                          PCI_CONFIG_SPACE ConfigSpace[],
                          ushort MaxNumOfFunctions,
                          ushort *NumOfFunctions)
```

Parameter

Parametername	Bedeutung
VendorID	vendor ID
DeviceID	device ID
ConfigSpace	Array von Config-Space
MaxNumOfFunctions	Max Anzahl von Einträgen im ConfigSpace[]
NumOfFunctions	Anzahl von gefundenen Funktionen [out Paramter]

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt
RM_BOUND_REACHED	Anzahl von MaxNumOfFunctions erreicht

Beschreibung

Scant den gesamten PCI-Bus über Bridges hinaus auf alle Funktionen, die der ausgewählten VendorID und DeviceID entsprechen. Um alle Devices am Bus zu suchen kann als VendorID = 0xFFFFH übergeben werden.

In das übergebene Array ConfigSpace[] werden die Inhalte des(der) gefundenen Device(s) eingetragen.

Die Devices werden dabei nach PCI-Konvention gefunden. Für SICOMP-IMC bedeutet das eine Zählweise von links nach rechts.

Die Funktion bricht die Suche bei vollem ConfigSpace Array ab.

Wird für die ausgewählte VendorID und DeviceID kein Device am Bus gefunden, dann wird NumOfFunctions = 0 und der Rückgabewert = RM_OK gesetzt.

Hinweis

Die Größe des Arrays ConfigSpace muß größer oder gleich von MaxNumOfFunctions sein..

3.1.2 RmPciSearchSubFunction

Funktion **Scant den PCI-Bus (in RMPCISCA.LIB enthalten)**

Syntax

```
#include <rmpci.h>
int RmPciSearchSubFunction ( ushort VendorID,
                             ushort DeviceID,
                             ushort SubVendorID,
                             ushort SubDeviceID,
                             PCI_CONFIG_SPACE ConfigSpace[],
                             ushort MaxNumOfFunctions,
                             ushort *NumOfFunctions)
```

Parameter

Parametername	Bedeutung
VendorID	vendor ID
DeviceID	device ID
SubVendorID	Subsystem vendor ID
SubDeviceID	Subsystem ID
ConfigSpace	Array von Config-Space
MaxNumOfFunctions	Max Anzahl von Einträgen im ConfigSpace[]
NumOfFunctions	Anzahl von gefundenen Funktionen [out Paramter]

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt
RM_BOUND_REACHED	Anzahl von MaxNumOfFunctions erreicht

Beschreibung

Scant den gesamten PCI-Bus über Bridges hinaus auf alle Funktionen, die der ausgewählten VendorID, DeviceID, Subsystem VendorID und Subsystem ID entsprechen.

In das übergebene Array ConfigSpace werden die Inhalte des(der) gefundenen Device(s) eingetragen.

Die Devices werden dabei nach PCI-Konvention gefunden. Für SICOMP-IMC bedeutet das eine Zählweise von links nach rechts.

Die Funktion bricht die Suche bei vollem ConfigSpace Array ab.

Wird für die ausgewählte VendorID und DeviceID kein Device am Bus gefunden, dann wird NumOfFunctions = 0 und der Rückgabewert = RM_OK gesetzt.

Hinweis

Die Größe des Arrays ConfigSpace muß größer oder gleich von MaxNumOfFunctions sein..

3.1.3 Aufbau des PCI-Config-Space

Der ConfigSpace ist gemäß PCI-Konvention wie folgt definiert:

```
uchar     PciBusNum       /* Nummer des Buses /
uchar     PciDeviceNum    /* Nummer des Devices /
uchar     PciFunctionNum  /* Nummer der Funktion */
ushort    VendorId
ushort    DeviceId
ushort    Cmdreg
ushort    StatReg
uchar     RevisionId
uchar     ClassCode[3]
uchar     CacheLineSize
uchar     LatencyTime
uchar     HeaderType
uchar     BIST
uint      BaseAdr0
uint      BaseAdr1
uint      BaseAdr2
uint      BaseAdr3
uint      BaseAdr4
uint      BaseAdr5
uint      CardbusCisPtr
ushort    SubsysVendorId
ushort    SubsysId
uint      ExpRomBaseAdr
uint      reserved[2]
uchar     IntLine         /* Interrupt – Nummer */
uchar     IntPin
uchar     Min_Gnt
uchar     Max_Lat
```

3.2 Interrupt Server

Der Interrupt Server kann statisch ins RMOS-System eingebunden werden, oder beim Booten nachgeladen werden. Der Interrupt Server darf aber nur einmal im System vorhanden sein.

RMISHSRV.LIB Statische Funktion

RMISHSRV.DRV Systemprogramm zum dynamischen Aufruf.

3.2.1 RmlShSrv

Funktion **Shared Interrupt Server für PCI-Bus**

Beschreibung Der Treiber initialisiert die PCI-Shared-Interrupt, trägt den Treiber in den RMOS-Katalog ein und setzt in einer Mailbox Zugriffsinformationen für den Client ab.

3.3 Interrupt Client RmIShCli

Die Funktionen des Interrupt-Clients ermöglichen einer RMOS-Task das API des Interrupt-Servers zu nutzen. Die Library RMISHCLI.LIB muss zu jeder RMOS-Task, die sie benutzt, dazugebunden werden.

Es wird zwischen 2 Modi des Interrupthandlings unterschieden:

- Im Mode 1 (Compatibility Mode) wird neben bereits vorhandenen I- und S-Handler-Routinen auch noch eine, vom Anwender zu schreibende, Routine GetIntReqState übergeben, anhand derer entschieden wird, ob ein aktiver Interrupt von den Handlern bearbeitet werden soll. Dadurch können bestehende I- und S-Handler-Routinen übernommen werden.

- Im schnelleren Mode 2 (Fast Mode) werden, wie mit der Standard RMOS-Funktion RmSetIntISHandler für flankengetriggerte Interrupts, dem Betriebssystem RMOS nur der I- und S-Handler-Routinen übergeben. Ob der aktive Interrupt für die Handler bestimmt ist, muss von der I-Handler-Routine selbst abgefragt werden.

3.3.1 RmInitShIntClient

Funktion **Initialisierung des Clients (in RMISHCLI.LIB enthalten)**

Syntax

```
#include <Rmpci.h>
int RmInitShIntClient( void )
```

Beschreibung Initialisierung des Clients, indem aus einer definierten Mailbox die Zugriffsinformationen auf den Server abgeholt wird. Erst dann können die anderen Funktionen des Clients ausgeführt werden.

Rückgabewert

RM_OK	Erfolgreiche Beendigung.
RM_IS_NOT_CATALOGED	Der SharedInterruptServer ist nicht katalogisiert

3.3.2 RmSetShIntISHandler1

Funktion **Shared Interrupt Handler im Compatibility-Mode installieren (in RMISHCLI.LIB enthalten)**

Syntax

```
#include <rmpci.h>
int RmSetShIntISHandler1 (uint  Irqx,
                          int _FIXED _FAR (*IHandlerEntry) (void),
                          int _FIXED _FAR (*SHandlerEntry) (void),
                          int _FIXED _FAR (*GetIntReqState) (void *),
                          void *UsrContext,
                          uint *chainID )
```

Parameter

Parametername	Bedeutung
Irqx	Interrupt-request Nummer (IRQ0-15) kann nach dem Funktionsaufruf RmPciSearchFunction (...) aus dem gefülltem Config-Space (IntLine) geholt werden.
IHandlerEntry	Pointer auf I-Handler, verantwortlich für Irqx; kann auch NULL sein, dann wird nichts eingetragen und der vorherige (evtl default) Handler bleibt bestehen.
SHandlerEntry	Pointer auf S-Handler, verantwortlich für Irqx kann auch NULL sein, dann wird nichts eingetragen und der vorherige (evtl default) Handler bleibt bestehen.
GetIntReqState	Pointer auf eine im Applikationstreiber zu implementierende Funktion. Der Returnwert dieser Funktion ist 0, wenn der Treiber nicht zuständig ist für den momentanen Interrupt. Der Returnwert ist 1, wenn der Treiber zuständig ist.
UsrContext	Pointer auf einen in der Applikation abgelegte Datenstruktur, zum Datenaustausch zwischen Applikation und Treiber. Dieser Zeiger wird jedesmal beim Aufruf von GetIntReqState mitgegeben, um dem Treiber einen Verweis auf seinen Kontext zu ermöglichen.
chainID	Pointer auf ID für Interrupt HandlerEntry [out Parameter] ID für InterruptHandlerEntry. Dies ist eine vom Shared-Interrupt-Server vergebene eindeutige ID, die der aufrufenden Applikation mitgeteilt wird. Die Applikation merkt sich diese ID, um bei Bedarf seinen Handler später mit der Funktion RmClrShIntISHandler wieder aus der verketteten Liste ausklinken zu können.

Beschreibung

Ruft eine Funktion des Servers auf, die einen Interrupt-Handler im Mode 1 (Compatibility Mode) im Server installierten. Dabei wird ein I-Handler, ein S-Handler und eine zugehörige GetIntReqState-Funktion für den vergegebenen IRQ installiert.

Diese Funktion wird beim Auftreten eines Interrupts vom Server aufgerufen und teilt dem Server mit, ob der Interrupt von diesem Treiber zu behandeln ist oder nicht. Dies kann festgestellt werden, indem die GetIntReqState-Function das Interrupt-Register überprüft, ob diese Baugruppe den Interrupt ausgelöst hat. Anschließend wird der I-Handler und/oder der S-Handler für diesen Treiber aufgerufen. Wie bei der Standard RMOS-Funktion RmSetIntISHandler wird der S-Handler nur aufgerufen, wenn der I-Handler mit einem Returnwert <> 0 beendet wird.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt
RM_OUT_OF_MEMORY	Nicht genügend Speicher verfügbar.
RM_INVALID_IRQ_NUMBER	Irqx ungültig
RM_INVALID_POINTER	Ungültiger Zeiger
RM_GOT_TIMEOUT	bei Zugriff auf verkettete Liste ist ein Timeout aufgetreten

3.3.3 RmSetShIntISHandler2

Funktion **Shared Interrupt Handler im Fast-Mode installieren (in RMISHCLI.LIB enthalten)**

Syntax

```
#include <rmpci.h>
int RmSetShIntISHandler1 (uint  Irqx,
                          int _FIXED _FAR (*IHandlerEntry) (void),
                          int _FIXED _FAR (*SHandlerEntry) (void),
                          void *UsrContext,
                          uint *chainID )
```

Parameter

Parametername	Bedeutung
Irqx	Interrupt-request Nummer (IRQ0-15) kann nach dem Funktionsaufruf RmPciSearchFunction (...) aus dem gefülltem Config-Space (IntLine) geholt werden.
IHandlerEntry	Pointer auf I-Handler, verantwortlich für Irqx; kann auch NULL sein, dann wird nichts eingetragen und der vorherige (evtl default) Handler bleibt bestehen
SHandlerEntry	Pointer auf S-Handler, verantwortlich für Irqx kann auch NULL sein, dann wird nichts eingetragen und der vorherige (evtl default) Handler bleibt bestehen.
UsrContext	Pointer auf eine in der Applikation abgelegte Datenstruktur, zum Datenaustausch zwischen Applikation und Treiber. Dieser Zeiger wird jedesmal beim Aufruf von GetIntReqState mitgegeben, um dem Treiber einen Verweis auf seinen Kontext zu ermöglichen.
ChainID	Pointer auf ID für Interrupt HandlerEntry [out Parameter] ID für InterruptHandlerEntry. Dies ist eine vom Shared-Interrupt-Server vergebene eindeutige ID, die der aufrufenden Applikation über die vorgegebene Adresse mitgeteilt wird. Die Applikation merkt sich diese ID, um bei Bedarf seinen Handler später mit der Funktion RmClrShIntISHandler wieder aus der verketteten Liste ausklinken zu können.

Beschreibung

Ruft eine Funktion des Servers auf, die einen Interrupt-Handler-Eintrag im Mode 2 (Fast Mode) im Server vornimmt. Dabei wird ein I-Handler und/oder ein S-Handler für den vergegebenen IRQ installiert.

Der I-Handler wird nach Auslösen des zugehörigen Interrupts aufgerufen. Er muß nun selbst prüfen, ob er für den Interrupt zuständig ist oder nicht und diesen bei Bedarf bearbeiten. Dies kann festgestellt werden, indem in der I-Handler-Routine das Interrupt-Register überprüft wird, ob diese Baugruppe den Interrupt ausgelöst hat. Falls er nicht zuständig ist, muß der Handler sofort mit return (2) und ohne Interruptquittierung verlassen werden. Falls der Handler den Interrupt zu bearbeitet hat, muß der Handler anschließend mit return (0) oder return (1) verlassen werden. Bei return (0) wird der S-Handler nicht aufgerufen, bei return (1) wird der S-Handler aufgerufen.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt
RM_OUT_OF_MEMORY	Nicht genügend Speicher verfügbar.
RM_INVALID_IRQ_NUMBER	Irqx ungültig
RM_INVALID_POINTER	Ungültiger Zeiger
RM_GOT_TIMEOUT	bei Zugriff auf verkettete Liste ist ein Timeout aufgetreten

3.3.4 RmClrShIntISHandler

Funktion **Shared Interrupt Handler entfernen (in RMISHCLI.LIB enthalten)**

Syntax

```
#include <rmpci.h>
int RmClrShIntISHander ( uint  Irqx,
                         uint chainID )
```

Parameter

Parametername	Bedeutung
Irqx	Interrupt-request Nummer (IRQ0-15) kann nach dem Funktionsaufruf RmPciSearchFunction (...) aus dem gefülltem Config-Space (IntLine) geholt werden.
chainID	ID für Interrupt-Handler

Beschreibung Ruft eine Funktion des Servers auf, die den entsprechenden Interrupt-Handler-Eintrag aus dem Server löscht, unabhängig vom Mode des Eintrages.

Wird der letzte Eintrag für diesen Interrupt entfernt, wird der default-Interrupt-Handler von RMOS installiert.

Rückgabewert

RM_OK	Funktion erfolgreich ausgeführt
RM_INVALID_SIZE	kein Handler zu löschen
RM_INVALID_ID	ungültige chainID
RM_INVALID_IRQ_NUMBER	Irqx ungültig

Abkürzungen / Begriffe

A

API

Application Programming Interface

Client

Verbraucher oder Kunde einer Dienstleistung, manchmal auch Bezeichnung eines Rollenverhaltens in der Beziehung zweier kooperierender Prozesse, in dem der Client die aktive, anfordernde Rolle annimmt.

Configuration Space

Jede Baugruppe am PCI-Bus muß einen Block von 64 Doppelworte für seine Konfiguration zur Verfügung stellen. Die ersten 16 Doppelworte sind durch die PCI-Spezifikation festgelegt.

Server

Dienstleistender Partner in der Beziehung zweier kooperierender Prozesse

Treiber

Programm-Modul im Betriebssystem, das die Bedienung bzw. Steuerung eines Peripherie-Bausteines übernimmt