**Wichtig - obwohl dieser Abschnitt bis Version 6.0.4 gilt, gibt es ab 6.0.5 einige wichtige Änderungen. Siehe nächste Abschnitte.**

Deklaration einer Interrupt-Prozedur.

Die möglichen Interrupt-Quellen hängen ganz vom Prozessor ab. Innerhalb einer CPU-Familie gibt es erhebliche Unterschiede. Interruptquellen werden in der Prozessor¬ beschreibungsdatei (xxx.dsc) deklariert.

Diese Prozedur setzt nur einen Eintrag in der Interrupt-Vektor-Tabelle und erzeugt einen Programmrahmen für die gewählten Registereinsparungen (siehe unten).

Außerdem müssen die E/A-Steuerregister der CPU für die spezifische Interrupt-Behandlung von der Anwendung selbst initialisiert werden (siehe die Beschreibungen im Handbuch der Steuerung) !

**Beispiel** (AVR Mega103):

**TIMER0** timer0 overflow interrupt

**TIMER0COMP** compare match interrupt timer0

**TIMER1** timer1 overflow interrupt

**TIMER1COMP** compare match interrupt timer 1

**TIMER1COMPA** compare match "a" interrupt timer 1

**TIMER1COMPB** compare match "b" interrupt timer 1

**TIMER1CAPT** capture event timer 1

**TIMER2** timer2 overflow interrupt

**TIMER2COMP** compare match interrupt timer2

**EERDY** eeprom ready

**ACOMP**  analog comparator

**SPIRDY**  SPI serial transfer complete

**ADCRDY**  ADC conversion complete

**INT0** External Interrupt 0

**INT1** External Interrupt 1

**INT2** External Interrupt 2

**INT3** External Interrupt 3

**INT4** External Interrupt 4

**INT5** External Interrupt 5

**INT6** External Interrupt 6

**INT7** External Interrupt 7

**RXRDY** uart1 rx complete

**UDRE** uart1 data register empty

Die Deklaration eines Interrupts erzeugt automatisch entsprechende Einträge in der Interrupt-Vektortabelle, sowie einen speziellen Code-Rahmen für die Interrupt-Prozedur.

Hier werden standardmäßig alle Register gespeichert -> vollständige Registerspeicherung.

Dies kann durch die Compilerschalter {$NOSAVE}, {$NOREGSAVE} und {$NOSHADOW} gesteuert werden.

Das globale Interrupt-Enable-Flag, das von der CPU automatisch zurückgesetzt wird, wird nicht verändert, d.h. die Interrupts bleiben zur Laufzeit deaktiviert. Wenn die 'end'-Anweisung erreicht wird, wird ein RETI (return from interrupt) ausgeführt, das den Interrupt aktiviert.

Das bedeutet, dass eine Interrupt-Prozedur so kurz wie möglich sein sollte, damit die anderen Interrupts nicht zu lange gesperrt sind.

Eine komplette Registerspeicherung benötigt etwa 20 Bytes im RAM, statisch, nicht auf Stack oder Frame. Aus diesem Grund ist ein gestapelter Interrupt nicht möglich. Das bedeutet, dass innerhalb einer Interrupt-Prozedur der Interrupt nie wieder freigegeben werden darf. Die CPU selbst macht diese Freigabe mit dem RETI-Befehl

Da beim XMega der globale Interrupt nicht beim Eintritt in eine Serviceroutine gesperrt wird, wird dies vom System selbst erledigt, ebenso die Freigabe. So ist es möglich, daß höher priorisierte Interrupts die Kontrolle übernehmen können. Dafür muss aber der globale Interrupt freigegeben werden. Dies kann erzwungen werden durch

 Definieren Sie Interruptible\_Ints = true;

in der Define-Sektion erzwungen werden. So können höher priorisierte Interrupts innerhalb des eigentlichen Interrupts die Kontrolle übernehmen. Systeminterne Interrupts laufen grundsätzlich mit einer Priorität von 2.

Wenn eine Unterbrechungsverschachtelung unverzichtbar ist, muss der Compilerschalter {$NOSAVE} verwendet werden. Damit werden nur die Merker und die Hauptarbeitsregister gespeichert. Die Prozeduren PushAllRegs und PopAllRegs können verwendet werden, um die restlichen Register über den Stack zu sichern, falls dies notwendig ist.

Die Verschachtelung von Interrupts ist sehr gefährlich und endet häufig mit einem Systemabsturz. Sie sollte daher nach Möglichkeit vermieden werden. Zumindest ist eine sehr sorgfältige Planung eine absolute Notwendigkeit.

Achtung!

Die korrekte Initialisierung von Interrupts erfordert in der Regel Einstellungen in verschiedenen Steuer- und Maskenregistern.

Bei den enthaltenen Treibern, die im Interruptmodus laufen, wird dies von den entsprechenden Treibern vorgenommen.

Definiert die Anwendung die Interrupts, kann der Compiler nicht weiterhelfen.

In diesen Fällen ist es die Aufgabe der Anwendung, die notwendigen Initialisierungen vorzunehmen !!

Wenn die Systemreaktion zu zeitkritisch ist und mehr Anweisungen ausgeführt werden müssen, empfiehlt sich folgendes Vorgehen:

Der Interrupt inkrementiert eine Semaphore und kehrt zurück. Ein Prozess wartet immer auf diese Semaphore und wird kurzzeitig die Kontrolle übernehmen.

***Interrupt*** *Int0;*

***begin***

*IncSema (sema0);*

***end;***

***Process*** *ProcessInt0 (32, 16 : iData);*

***begin***

*WaitSema (sema0); {wait for sema0 > 0 }*

 *...*

***end;***

Interrupt Service Routinen können eigene lokale Variablen haben. Da es hier keinen Frame gibt, sind diese Variablen statisch/nichtflüchtig, d.h. sie werden im iData (SRAM)-Bereich abgelegt und sind immer zugänglich

(statisch) und ihr Inhalt ist immer gültig (nicht-flüchtig).

Innerhalb der Interrupt-Prozedur kann auf sie wie auf jede andere Variable zugegriffen werden, und aufgrund ihrer statischen Natur können sie auch von anderen Teilen der Anwendung verwendet werden. Da sie aber eine Eigenschaft einer Prozedur sind, muss ein Zugriff qualifiziert werden:

***Interrupt*** *Timer1;*

***var*** *abc : byte;*

***begin***

 *abc:= 123; // no qualification*

 *...*

***end****;*

*...*

*Interrupt\_Timer1.abc:= $67; // qualify with "Interrupt\_name"*

Bitte beachten Sie, dass „Interrupt\_“ immer vor dem Qualifier stehen muss.

Grundsätzlich werden alle Interrupts, die nicht von einem Interrupt-Handler unterstützt werden, von einer Standard/Dummy-Interrupt-Service-Routine mit einem einfachen „RETI“ behandelt. Zu Debug- und Testzwecken kann dieser Interrupt-Fehler von der Anwendung unterstützt werden. Normalerweise sollte eine solche Bedingung nie auftreten, so dass diese Routine nie von Interrupts aufgerufen werden sollte. Die Implementierung muss in der Anwendung vorgenommen werden:

***Interrupt*** *IntErrorHandler;*

***begin***

 *...*

***end****;*

#### Push, Pop

***Procedure*** *Push (regnum : byte | regname : internal);* //e.g. Push (24);

***Procedure*** *Pop (regnum : byte | regname : internal);* //e.g. Pop (\_ACCFLO);

Für eine bessere Lesbarkeit. Erzeugt den gleichen Code wie

ASM: PUSH ... ; or

ASM: POP … ;

#### PushRegs, PopRegs

***Procedure*** *PushRegs;* // working registers to stack

***Procedure*** *PopRegs;* // working registers from stack

Diese Prozeduren sind vereinfachte Versionen von PushAllRegs und PopAllRegs und können immer dann als Paare in Interrupts verwendet werden, wenn nur die 4 wichtigen Register (ACCA, ACCB usw.) gespeichert werden.

2.1.1.2 PushAllRegs, PopAllRegs

Manchmal, aber nur manchmal, ist es sinnvoll, globale Interrupts in einer Interrupt-Prozedur wieder zu aktivieren, z.B. einen Timer-Interrupt. Dies vermeidet zu lange Interrupt-Sperrzeiten. Wenn in diesem Fall Register gesichert werden müssen, kann dies nicht mit der üblichen automatischen Registersicherung (Schalter $NOSHADOW inaktiv) geschehen.

Diese Registerspeicherung unterstützt keine „verschachtelten Interrupts“.

Um alle Register in einer solchen Funktion zu speichern, in der der globale Interrupt wieder aktiviert werden muss, muss die Registerspeicherung auf besondere Weise erfolgen:

*{$NoSave}*

***Interrupt*** *TIMER1COMPA; // TickTimer*

***begin***

 *PushAllRegs;*

 *EnableInts;*

 *.*

 *...*

 *PopAllRegs;*

***end****;*

Der Schalter {$NoSave} ist hier zwingend erforderlich.

Diese Methode sollte nur verwendet werden, wenn die globalen Interrupts in einer Interrupt-Service-Routine aktiviert werden müssen. Der Benutzer sollte genau wissen, warum und was er hier tut :-)

2.1.2 Interrupts und der Optimierer

Bei der Verwendung des Optimierers gelten viele der oben genannten Regeln nicht. Stattdessen berechnet der Optimierer die vom Interrupt benötigten Register und fügt die notwendigen Push- und Popsignale ein. Dies ist effizienter als SaveAllRegs. Daher sollte bei der Verwendung des Optimierers immer {$NOSAVE} verwendet werden. Andernfalls werden die Register zweimal gespeichert.

2.1.3 Unterbrechungsalias und Zusammenführung

Es ist möglich, eigene Namen für Interrupts zu definieren. Die Methode dazu ist die Definition einer Interrupt-Konstante wie die folgende

const

 MyInt : Interrupt = 17;

Damit erhält der Interrupt Nummer 17 den Alias MyInt. Nun können Sie MyInt oder den ursprünglichen Namen als Interrupt verwenden. Wenn beide verwendet werden, werden die beiden Definitionen zusammengeführt. Die Reihenfolge, in der sie ausgeführt werden, ist nicht festgelegt.

Mehrere Aliase können auf denselben Interrupt verweisen und werden alle zusammengeführt.Verwenden Sie diese Möglichkeit natürlich mit äußerster Vorsicht.