Kapitel 13

Interrupts

Interrupts

Problem : Während der Ausführung eines Programms tritt ein Ereignis ein, auf das das System unmittelbar mit der Ausführung einer bestimmten Programmroutine reagieren muss.

Beispiel : Der Benutzer bewegt die Maus an seinem PC – der Mauszeiger auf dem Bildschirm muss entsprechend verändert werden.

Mögliche **Lösungen** für das Beispiel :

1. **Polling**

   Das laufende Programm fragt in regelmäßigen Abständen den Mauscontroller, ob die Maus bewegt wurde und ruft ggf. die Routine für die Steuerung des Mauszeigers auf.

2. **Interrupt**

   Der Mauscontroller meldet die Bewegung der Maus an das System. Das System unterbricht bei der nächsten passenden Gelegenheit die Ausführung des laufenden Programms und startet die Routine für die Steuerung des Mauszeigers

- Polling ist nur bei speziellen Szenarien praktikabel.
  - z.B. beim Testen von E/A-Geräte-Anschlüssen.
- Interrupts sind die bewährte Methode
  - Viele Ereignisse mit zum Teil hoher Wiederholungsrate müssen verarbeitet werden. Mit Polling hätten die Abfragen auf Vorliegen eines Ereignisses einen immensen Performanceverlust zu Folge.
  - Die Anwendungsprogramme und die Interrupt Service Routinen sind entkoppelt.
  - Ein gewisser Hardwareaufwand ist allerdings erforderlich.

Interrupt-Klassen

- E/A Interrupts (I/O interrupts)
- Timer Interrupts
- Maschinenfehler Interrupts (machine check interrupts) – asynchron zur Programmausführung
  - z.B.: Hardwarefehler
- Interrupts wegen Ausnahmesituationen bei der Programmausführung (program interrupts) - synchron zur Programmausführung
  - z.B.: Division durch 0; ungültiger OpCode, ungültige Adresse, Page Fault
- Software Interrupts
  - z.B.: x86 : INT Befehl; /390 : Supervisor Call (SVC); 8051 : Interrupt Flag setzen

- Im Prinzip handelt es sich bei einem Interrupt um einen Unterprogrammaufruf,
  - der von der **Hardware initiiert** wird, und
  - der von einer **fest vorgegebenen Adresse** startet.

Interrupt Mechanismus

- Das Kontrollprogramm des Systems muss Einstellungen so vornehmen, dass Interrupts für die Interruptklassen, die bedient werden sollen, zugelassen sind. Man sagt, **Interrupts werden enabled**.
- Eine Prozessorkomponente oder eine periphere Einheit meldet eine Interruptanforderung (**Interrupt Request**) an.
- Falls der anliegende Interrupt Request zugelassen (enabled) ist, unterbricht der Prozessor am Ende des gerade ausgeführten Befehls das laufende Programm und rettet den Zustand des unterbrochenen Programms auf dem Stack.
  - Im Fall des 8051 Prozessors wird lediglich der Inhalt des Befehlszählers (PC) auf den Stack gerettet.
- Dann startet der Prozessor die **Interrupt Service Routine (ISR)** für diese Interruptklasse. Die Startadressen für die Interrupt Service Routinen sind fest vorgegeben (siehe nächste Folie).
- Um zu verhindern, dass eine laufende Interrupt Service Routine ihrerseits unterbrochen wird, sperrt der Prozessor die Interrupt Requests (Interrupts werden **intern** disabled).
- Die Interrupt Service Routine rettet die Inhalte aller Resourcen, die sie selbst auch verwendet, auf den Stack (z.B. Akkumulator, PSW, Register, ...).
- Die Interrupt Service Routine führt die notwendigen Operationen durch.
- Die Interrupt Service Routine stellt die zuvor auf den Stack geretteten Inhalte zurück.
- Die Interrupt Service Routine beendet sich selbst durch die Ausführung eines Return-from-Interrupt Befehls.
  - Beim 8051 Prozessor ist dies der **RETI** Befehl.
- Der Prozessor führt diesen Befehl aus, indem er den Zustand des unterbrochenen Programms vom Stack holt und wieder aktiviert.
- Außerdem ändert der Prozessor die internen Einstellungen so, das Interrupts wieder zugelassen sind (intern disabled Interrupts werden wieder enabled).

Startadressen für 8051 Interrupt Service Routinen

Table 7-2
Interrupt Source and Vectors

Interrupt Source	Interrupt Vector Address	Interrupt Request Flags
External Interrupt 0	0003_H	IE0
Timer 0 Overflow	$000B_H$	TF0
External Interrupt 1	0013_H	IE1
Timer 1 Overflow	$001B_H$	TF1
Serial Channel	0023_H	RI / TI
Timer 2 Overflow / Ext. Reload	$002B_H$	TF2 / EXF2
A/D Converter	0043_H	IADC
External Interrupt 2	$004B_H$	IEX2
External Interrupt 3	0053_H	IEX3
External Interrupt 4	$005B_H$	IEX4
External Interrupt 5	0063_H	IEX5
External Interrupt 6	$006B_H$	IEX6
Wake-up from power-down mode	$007B_H$	–
CAN controller	$008B_H$	–
External Interrupt 7	$00A3_H$	–
External Interrupt 8	$00AB_H$	–
SSC interface	0093_H	TC / WCOL

Enable/Disable Interrupts per Programm

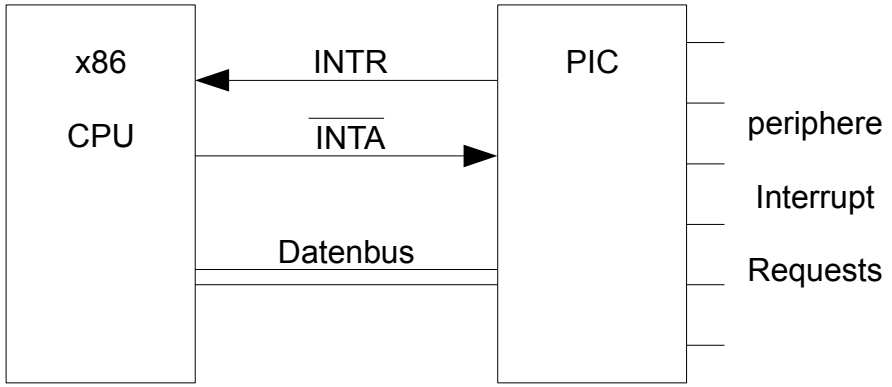
- Interrupts können einzeln enabled/disabled werden :
  - Beim 8051 durch Setzen/Löschen der entsprechenden **Interrupt Enable Flags** in SFRs.
- Alle Interrupts können auch kollektiv enabled/disabled werden (um z.B. eine Befehlssequenz auszuführen, die nicht unterbrochen werden darf) :
  - x86 Befehle : STI (Set Interrupt Enable); CLI (Clear Interrupt Enable)
  - 8051 Befehle : SETB EA; CLR EA (SFR Bit EA : Enable All Interrupts – IEN0.7)
- Ein Interrupt wird nur ausgeführt, wenn sowohl sein individuelles Enable Flag als auch das kollektive Enable Flag des Systems gesetzt sind.
- Einige Systeme besitzen einen Nicht-Maskierbaren Interrupt (NMI), der nicht disabled werden kann. Er wird in extremen Situationen (z.B. Spannungsverlust) aktiviert. Beim 8051 gibt es keinen NMI.

Interruptsteuerung

- Bei **x86 Systemen** ist die Behandlung der externen Interrupts in einen eigenen Prozessor, den **Peripheren Interrupt Controller (PIC)** ausgelagert.
  - Der PIC hat Eingangsleitungen für alle möglichen Interruptquellen.
  - Der PIC meldet den Interrupttyp über einen Adresszeiger oder eine Kennzahl an den Prozessor.
  - Der Prozessor ermittelt die Startadresse der Interrupt Service Routine (ISR) aus einer Adresstabelle im Speicher mit dem Adresszeiger als Index.
- Bei **8051 Systemen** werden die Interruptanforderungsleitungen direkt an Prozessorpins geführt, und die Behandlung der Interrupts findet im Interruptcontroller statt. Dieser ist Teil des 8051 Prozessors.
  - Die Startadressen der Interrupt Service Routinen sind in der Hardware verdrahtet.

x86 Interrupt Mechanismus

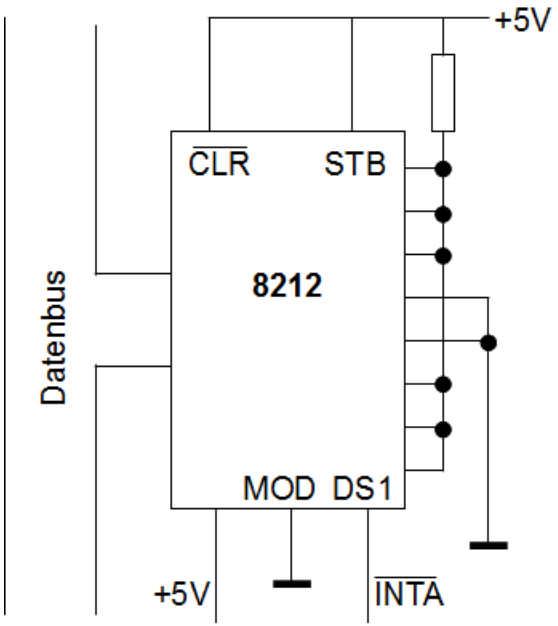
- Der x86 Prozessor weist zwei zusätzliche Steuerleitungen für das Signalisieren von Interrupts auf : Interrupt Request Leitung (**INTR**) und die Interrupt Acknowledge Leitung (**$\overline{\text{INTA}}$**).
- Die peripheren Einheiten melden ihre Interrupt Requests am PIC an.
- Der PIC legt im Fall von mehreren anstehenden peripheren Interrupt Requests auf Grund von Prioritätsspezifikationen die Reihenfolge fest, in der die Interrupt Requests dem x86 Prozessor gemeldet werden.
- Um einen Interrupt Request anzumelden, aktiviert der PIC die INTR Steuerleitung.



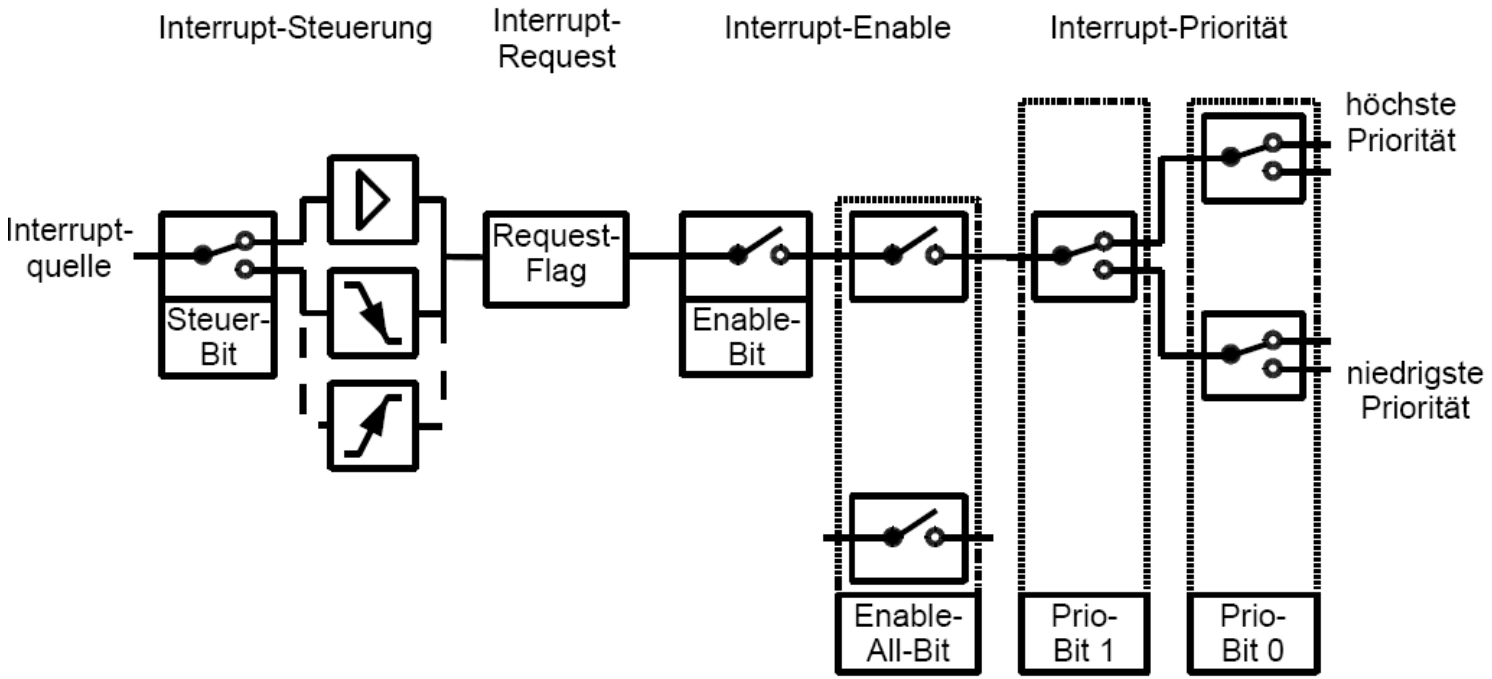
- Wenn der x86 Prozessor für die Behandlung des Interrupt Requests bereit ist (Interrupts sind enabled, keine andere ISR gerade aktiv, …), aktiviert er die $\overline{\text{INTA}}$ Steuerleitung.
- Darauf stellt der PIC eine Interrupt-Nummer auf den Datenbus und deaktiviert die INTR Steuerleitung wieder. Die Interrupt-Nummer identifiziert den Interrupt Request, der als nächstes zu bedienen ist.
- Der Prozessor übernimmt die Interrupt-Nummer und löscht das $\overline{\text{INTA}}$ Steuersignal wieder.
- Jetzt kann der PIC erneut die INTR Leitung aktivieren.
- Im x86 Speicher existiert eine Liste aller ISR-Einsprungadressen. Jedes System ist damit individuell konfigurierbar.
- Aus der Interrupt-Nummer auf dem Datenbus ermittelt der x86 Prozessor den zutreffenden Eintrag und startet die ISR an dieser Adresse.
- Den x86 Mechanismus bezeichnet man auch als **vektorisierten** Interrupt-Mechanismus, weil es einen Vektor von ISR Startadressen gibt.

Beispiel einer einfachen Schaltung eines Interrupt Request Identifiers

- 8212 Portbaustein im Input Mode (MOD = 0).
- Enthält ein Flipflop und einen Tri-State-Treiber pro Bit.
- Wegen STB = 1 wird der Inputwert sofort in die Flipflops übernommen.
- Aktivierung von DS1 aktiviert die Tri-State-Treiber,
  d.h. $\overline{\text{INTA}}$ schaltet den Wert in den Flipflops auf den Datenbus.
- Mit der ansteigenden Flanke von $\overline{\text{INTA}}$ werden die Tri-State-Treiber wieder deaktiviert.



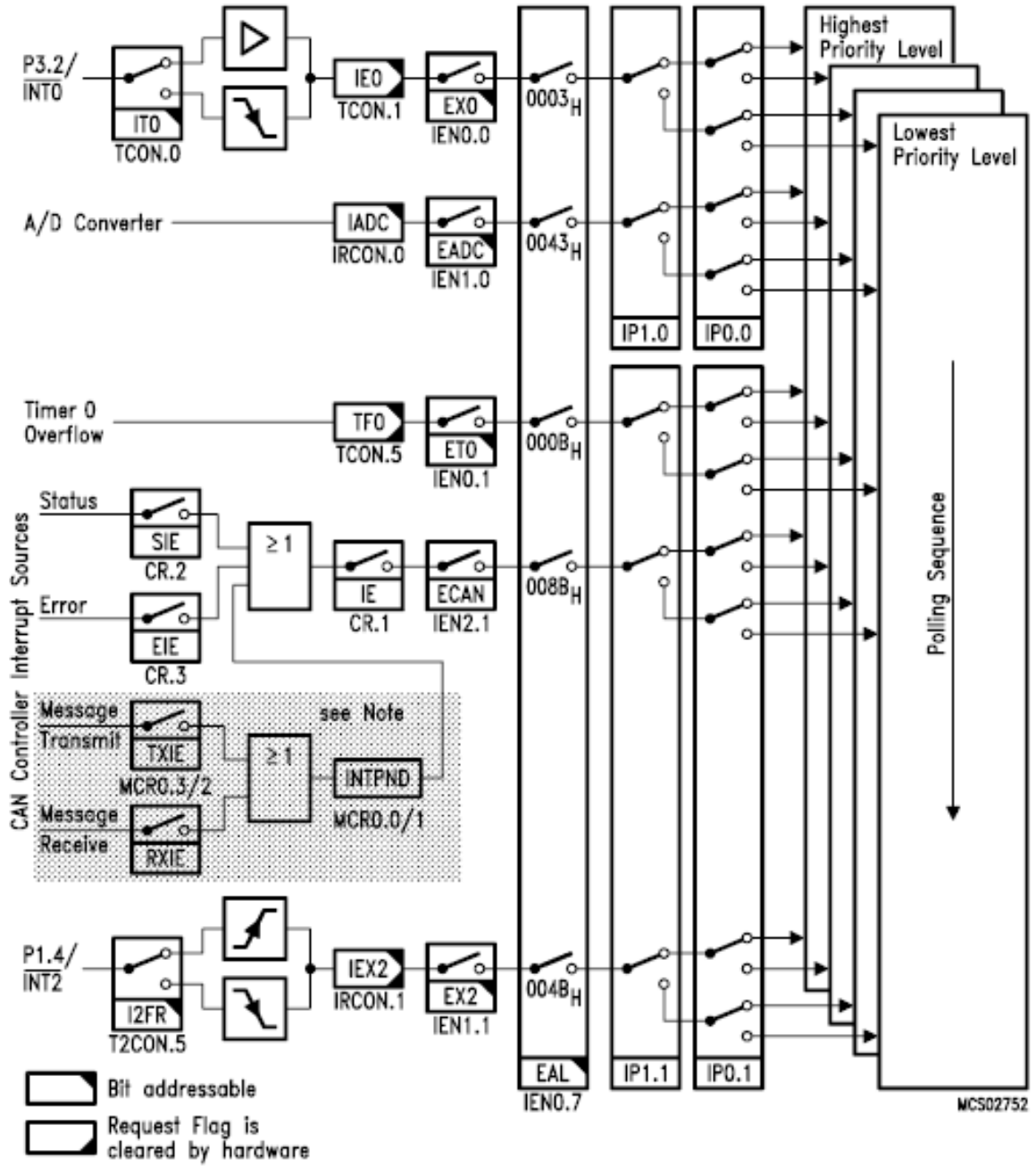
Behandlung von Interruptsignalen im 8051 Prozessor



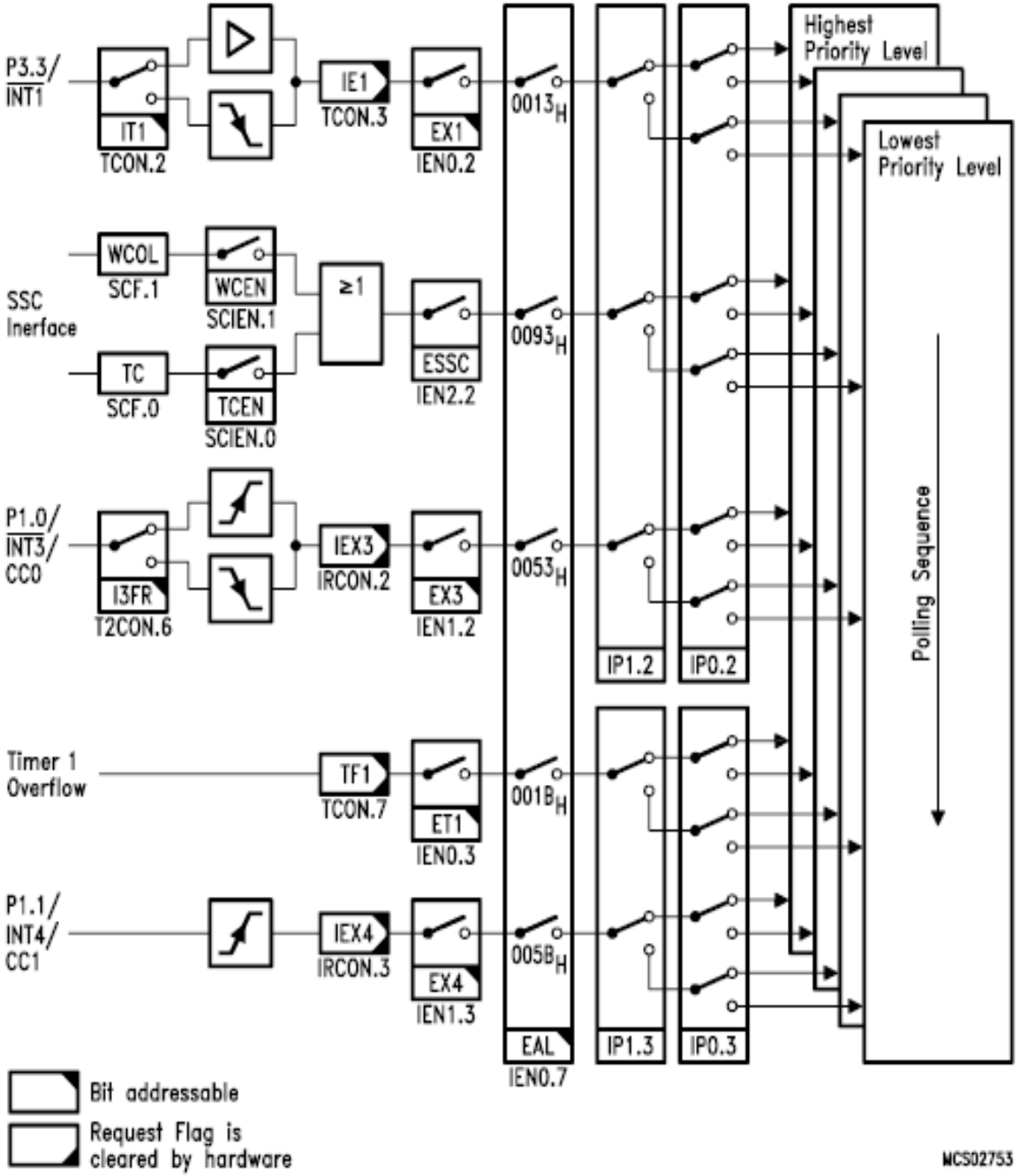
- Durch entsprechendes Setzen des Steuerbits kann ausgewählt werden, ob der Interrupt Request bei ansteigender oder abfallender Flanke oder bei aktivem Pegel des Interrupt Signals gesetzt werden soll.
- Das Enable Bit erlaubt oder verhindert die Weitergabe genau dieses Interrupt Requests in den Interrupt Controller.
- Das Enable-All Bit erlaubt oder verhindert die Weitergabe aller Interrupt Requests in den Interrupt Controller.
- Durch entsprechendes Setzen der Prioritäts Bits wird dem Interrupt eine von 4 Prioritätsstufen zugewiesen.

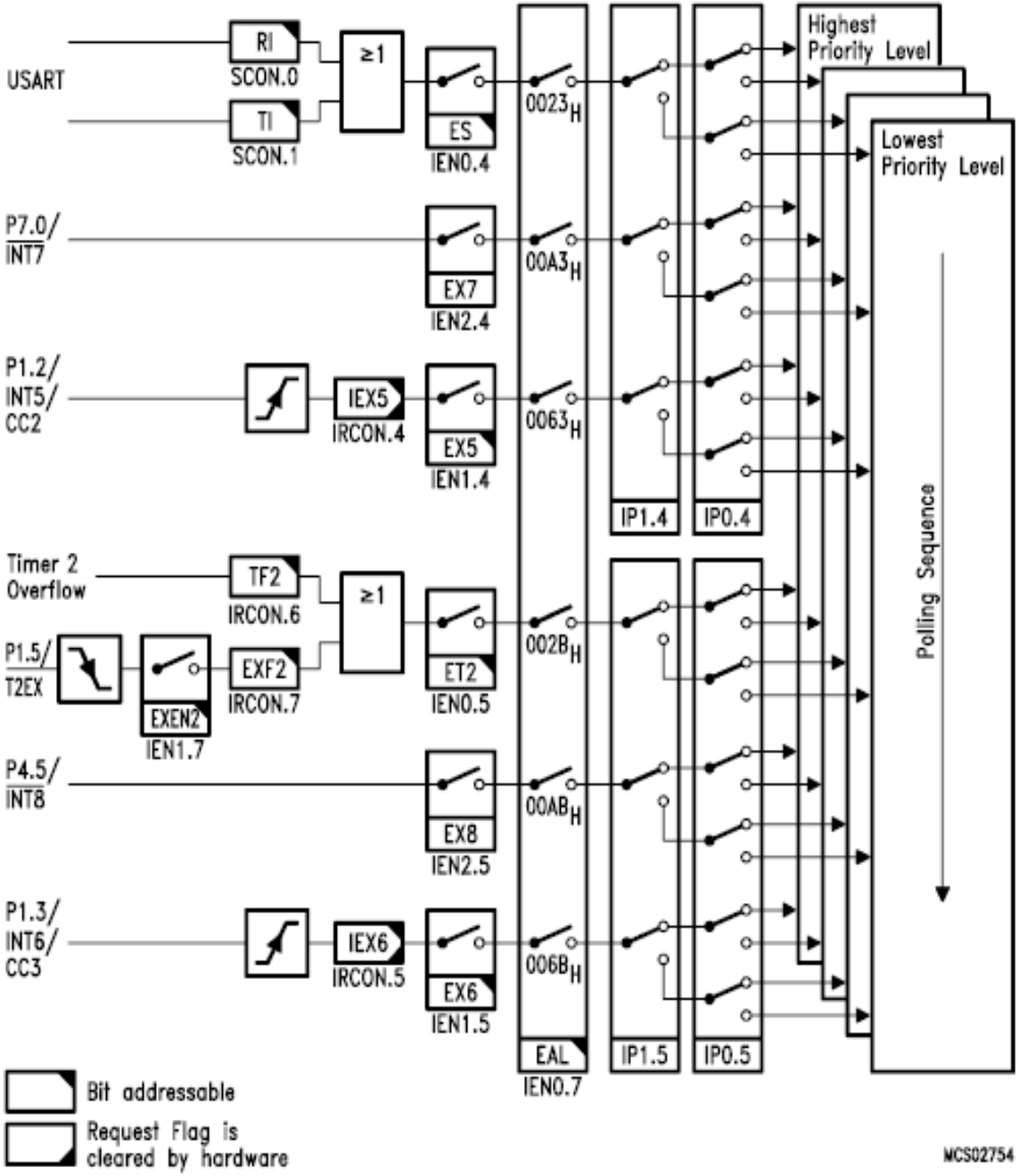
8051 Interruptmechanismus :

Die folgenden drei Bilder **Figure 7-1, 7-2 und 7-3 : Interrupt Structure Overview** im 8051 User's Manual stellen zusammen den 8051 Interruptcontroller dar.



Note: Each of the 15 CAN controller message objects provides the bits/flags in the shaded area.


P3.3/
INT1
IT1
TCON.2
IE1
TCON.3
EX1
IEN0.2
0013H
Highest
Priority Level
Lowest
Priority Level
SSC
Inerface
WCOL
SCF.1
WCEN
SCIEN.1
TC
SCF.0
TCEN
SCIEN.0
≥1
ESSC
IEN2.2
0093H
P1.0/
INT3/
CC0
I3FR
T2CON.6
IEX3
IRCON.2
EX3
IEN1.2
0053H
IP1.2
IP0.2
Polling Sequence
Timer 1
Overflow
TF1
TCON.7
ET1
IEN0.3
001BH
P1.1/
INT4/
CC1
IEX4
IRCON.3
EX4
IEN1.3
005BH
EAL
IEN0.7
IP1.3
IP0.3
Bit addressable
Request Flag is
cleared by hardware
MCS02753


USART
RI
SCON.0
TI
SCON.1
≥1
ES
IEN0.4
0023H
P7.0/
INT7
EX7
IEN2.4
00A3H
P1.2/
INT5/
CC2
IEX5
IRCON.4
EX5
IEN1.4
0063H
Timer 2
Overflow
TF2
IRCON.6
P1.5/
T2EX
EXEN2
IEN1.7
EXF2
IRCON.7
ET2
IEN0.5
002BH
P4.5/
INT8
EX8
IEN2.5
00ABH
P1.3/
INT6/
CC3
IEX6
IRCON.5
EX6
IEN1.5
006BH
EAL
IEN0.7
IP1.4
IP0.4
IP1.5
IP0.5
Highest
Priority Level
Lowest
Priority Level
Polling Sequence
Bit addressable
Request Flag is
cleared by hardware
MCS02754

Interruptquellen

- Externe und interne Interruptquellen
  - **Externe** Interruptquellen INT0 ... INT8 signalisieren Interruptanforderungen über ausgewählte Port Pins.
    - Interrupts werden über die Alternativen Port Funktionen gemeldet.
  - **Interne** Interruptquellen signalisieren Interruptanforderungen von integrierten Komponenten
    - A/D Converter
    - Timer 0, Timer 1 und Timer 2
    - CAN Controller
    - SSC
    - USART (serielle Schnittstelle)
- Löschen der Interrupt Request Flags :
  - Einige Interrupt Request Flags werden von der Hardware zurückgesetzt, wenn die ISR gestartet wird. (siehe Markierung in den Interrupt Request Flag Darstellungen der Interrupt Controller Bilder).
  - Die anderen Request Flags müssen von der ISR zurückgesetzt werden.
- Einige Komponenten senden ihre Interruptanforderungen **direkt** in den Interruptcontroller.
  - z.B. CAN Bus Interrupt, SSC Interrupt, INT7, INT8
  - Die ISR muss dafür sorgen, dass die Interruptquelle zurückgesetzt wird. Sonst wird nach Ausführung des RETI Befehls sofort wieder ein neuer Interrupt Request ausgelöst.
- Interrupts können einzeln enabled/disabled werden über die Bits in den SFR Lokationen IEN0, IEN1 und IEN2.
- Alle Interrupts können zusätzlich **global** enabled/disabled werden über das Bit EAL (SFR Lokation IEN0.7).

Interrupt-Durchführung

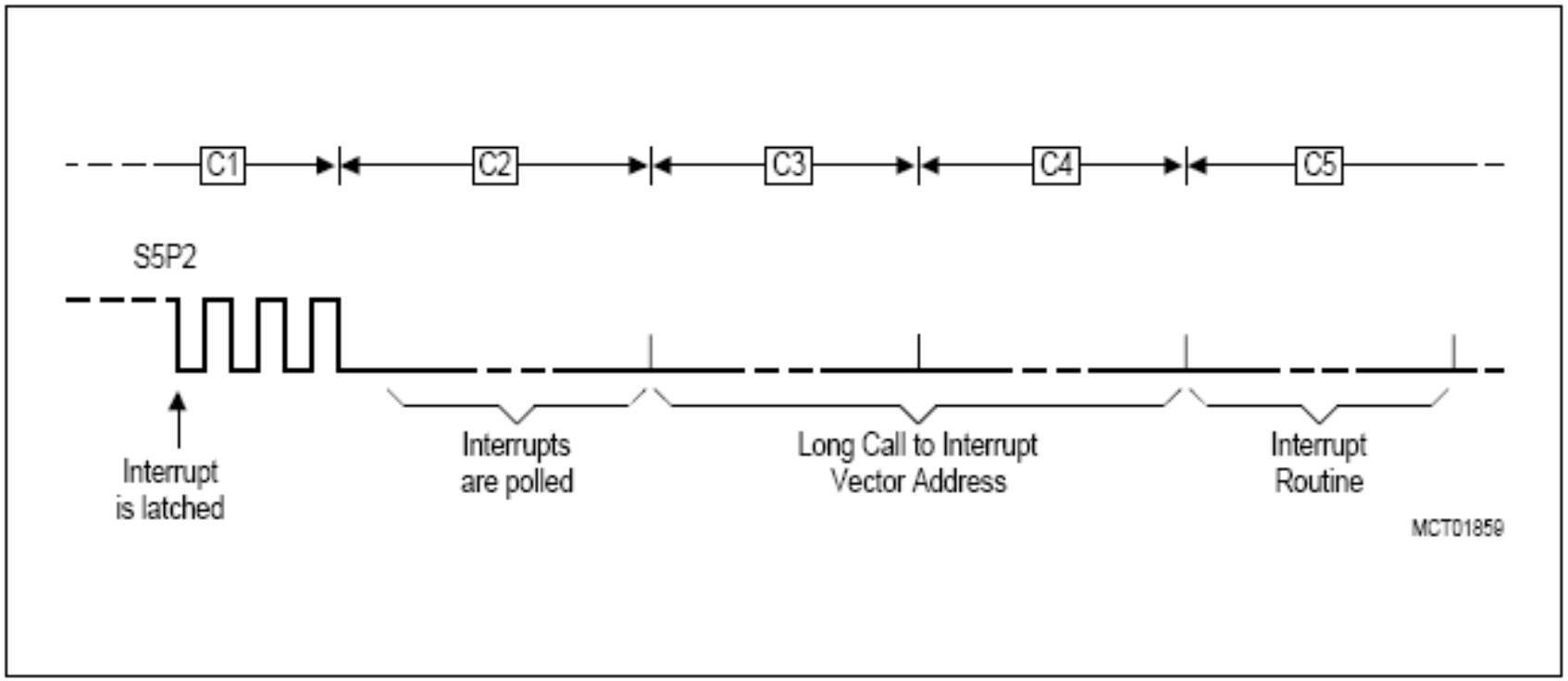


Figure 7-4
Interrupt Response Timing Diagram

- In jedem Maschinenzyklus (in S5P2) werden die Interrupt Requests intern gespeichert.
- Im darauf folgenden Maschinenzyklus wird untersucht, ob einer der gespeicherten Interrupt Requests aktiviert ist.
  - Die Interrupt Requests werden in der Reihenfolge ihrer Prioritäten untersucht (Interrupt Polling).
- Muss ein Interrupt durchgeführt werden, erfolgt in den nächsten beiden Maschinenzyklen die Unterbrechung mit dem Sprung zum Beginn der zugehörigen ISR.

Interrupt-Durchführung (Forts.)

- Der Start der ISR nach dem Aktivieren der Interrupt Requests kann sich noch weiter als die o.g. vier Maschinenzyklen (C1 … C4) verzögern,
  - wenn eine ISR mit höherer Priorität gerade ausgeführt wird oder eine Interruptanforderung mit höherer Priorität noch ansteht.
  - wenn die Befehle, in denen die Interruptanforderungen gespeichert (C1) oder untersucht (C2) werden länger als einen Maschinenzyklus dauern.
  - wenn gerade ein RETI Befehl ausgeführt wird.
    - Nach dem RETI Befehl wird noch ein Befehl im Hauptprogramm ausgeführt.
  - wenn ein Zugriff auf ein Interrupt-Enable-Flag (IEx) oder ein Interrupt-Prioritäts-Flag (IPx) erfolgt.
    - Diese Befehle lassen nicht zu, dass ein Interrupt prozessiert wird.

Interrupt-Prioritäten

- Die Bits in den SFR Lokationen IP0 und IP1 dienen der **expliziten Prioritätssteuerung** der Interrupts.

	MSB							LSB	
Bit No.	7	6	5	4	3	2	1	0	
$A9_H$	OWDS	WDTS	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0	IP0
Bit No.	7	6	5	4	3	2	1	0	
$B9_H$	PDIR	–	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0	IP1

- Standardmäßig (ohne dass eines der 12 Prioritätsbits gesetzt ist) ist bereits eine bestimmte Priorität vorgegeben.
  - Die Interrupts sind in 6 Gruppen zu je zwei oder drei Interrupts zusammengefasst (die 6 Zeilen in Tab. 7).
    - z.B.: Die Gruppe 1 enthält den INT0 und den A/D Converter Interrupt.

Table 7-1
Interrupt Source Structure

Interrupt Group	Associated Interrupts High Priority → Low Priority			Priority
1	External interrupt 0	–	A/D converter interrupt	High
2	Timer 0 overflow	CAN controller interrupt	External interrupt 2	↓
3	External interrupt 1	SSC interrupt	External interrupt 3	
4	Timer 1 overflow	–	External interrupt 4	
5	Serial channel interrupt	External interrupt 7	External interrupt 5	
6	Timer 2 interrupt	External interrupt 8	External interrupt 6	Low

Interrupt-Prioritäten (Forts.)

- Alle Interrupts einer Gruppe haben Vorrang vor allen Interrupts der niedrigeren Gruppen.
  - Stehen gleichzeitig zwei Interrupts für verschiedene Prioritätsgruppen an, wird zuerst der Interrupt für die weiter oben stehende Gruppe verarbeitet.
  - Wird ein Interrupt Request in einer Gruppe aktiviert, während ein Interrupt einer niedrigeren Gruppe verarbeitet wird, so wird die ISR des Interrupts der niedrigeren Gruppe unterbrochen, und die ISR für den Interrupt der höheren Gruppe wird ausgeführt.
    - Auf diese Weise können im Extremfall 6 ISRs geschachtelt sein.
- Stehen gleichzeitig zwei Interrupts für die selbe Prioritätsgruppe an, wird zuerst der Interrupt verarbeitet, der innerhalb der Gruppe weiter links steht.
  - Ein Interrupt **unterbricht** die laufende ISR eines anderen Interrupts der selben Gruppe **nicht**.
- Mit den Bits 0 ... 6 in den SFR Lokationen IP0 und IP1 wird jeder Gruppe explizit eine von vier Prioritätsebenen zugewiesen, die mit Vorrang vor den o.g. eingebauten Prioritäten behandelt wird.
  - Mit den Bits in IP0 und IP1 kann im Prinzip eine vom Standard abweichende Prioritätsreihenfolge der Gruppen eingestellt werden.

Programmierung des Interruptcontrollers

- Die Bits zur Steuerung der Interruptsignale, die Interrupt Request Flags, die Bits zur Prioritätssteuerung sind alle über SFR-Adressen erreichbar und können somit auch per Programm ausgelesen und verändert werden.
  - Viele dieser Bits sind sogar bitadressierbar und somit mit einfachen Befehlen wie SETB und CLR zu manipulieren.
  - Auf diese Weise können Interrupt Request Flags gesetzt werden, um einen Interrupt per Programm auszulösen.
  - Interrupt Request Flags können auch zurück gesetzt werden, um gemeldete Interrupt Requests zu unterdrücken (solange die zugehörige ISR noch nicht gestartet wurde).
- Eine Besonderheit weisen INT0 und INT1 auf, wenn sie von einem **pegelgesteuerten** Signal geschaltet werden. In diesem Fall folgen die Request Flags IE0 und IE1 dem am Portpin anliegenden Wert.
  - Wenn der Wert eines solchen Request Flags durch einen Befehl verändert wird, bleibt der Befehl unwirksam, da der Wert des Request Flags durch den Pegel auf dem Portpin sofort wieder auf den alten Wert zurückgesetzt wird.
  - Die ISR muss dafür sorgen, dass die Interruptquelle in der peripheren Einheit zurückgesetzt wird. Sonst wird nach der Ausführung des RETI Befehls sofort wieder ein Interrupt ausgelöst.
- Das (zeitweise) Deaktivieren des globalen Interrupt Enable Flags EA verhindert, dass Interrupts prozessiert werden. In einer solchen Phase kann z.B. im Polling Mode gearbeitet werden.
- Die Einstellungen des Interruptcontrollers sollte während der Programminitialisierung erfolgen und im laufenden Programm möglichst nicht mehr verändert werden.
  - Änderungen der Einstellungen im laufenden Betrieb könnten unvorhersehbare Folgen haben.
  - Die letzte Einstellungsaktion ist das globale Zulassen der Interrupts (SETB EA).