Nebenläufigkeit

Unterbrechungen

Binding Rechnersystem-Umwelt

- Es muss ein Mechanismus gefunden werden, der es erlaubt, Änderungen der Umgebung (z.B. Druck einer Taste) zu registrieren.
- **1. Ansatz**: Abfrage (Polling)
  Es werden die E/A-Register reihum nach Änderungen abgefragt und bei Änderungen spezielle Antwortprogramme ausgeführt.
  - Vorteile:
    - bei wenigen EA-Registern sehr kurze Latenzzeiten
    - bei einer unerwarteten Ereignisflut wird das Zeitverhalten des Programms nicht übermäßig beeinflusst
    - Kommunikation erfolgt synchron mit der Programmausführung
  - Nachteile:
    - die meisten Anfragen sind unnötig
    - hohe Prozessorbelastung
    - Reaktionszeit steigt mit der Anzahl an Ereignisquellen

Lösung: Einführung des Begriffs der Unterbrechung

- **2. Ansatz**: Unterbrechung (Interrupt)
- Eine Unterbrechung stoppt die Verarbeitung des laufenden Programms. Die Wichtigkeit des Ereignisses, welches die Unterbrechung ausgelöst hat, wird überprüft. Darauf basierend erfolgt die Entscheidung, welche Reaktion erfolgt.
- Vorteile:
  - Prozessorressourcen werden nur dann beansprucht, wenn es nötig ist
- Nachteile:
  - Nicht-Determinismus: Unterbrechungen asynchron zum Programmlauf (und zum Prozessorzustand) eintreffen.

Unterbrechungen

- **Unterbrechungen**: Stopp des Hauptprogrammablaufs, Aufnahme der Programmausführung eines „Unterbrechungsbehandlers (UBB)“ an einer anderen Stelle; nach Beendigung des UBB (zumeist) Rückkehr an die Stelle des Auftritts der Unterbrechung im Hauptprogramm.
- **Synchrone** Unterbrechungen: treten, falls sie auftreten, immer an *derselben Stelle* im Programmcode auf. Man bezeichnet sie auch als *Traps* oder *Exceptions* bzw. „Software-Interrupts“
- **Asynchrone** Unterbrechungen: Auftrittszeitpunkt ist unbestimmt; es kann nicht gesagt werden, an welcher Stelle der Hauptprogrammausführung der Prozessor zum Zeitpunkt der Unterbrechung ist. Asynchrone Unterbrechungen werden auch als Interrupts bezeichnet; weil sie von der Hardware-Peripherie erzeugt werden, auch als *Hardware-Interrupts*. Sie üben „Brückenfunktion“ zwischen Hardware und Software aus.

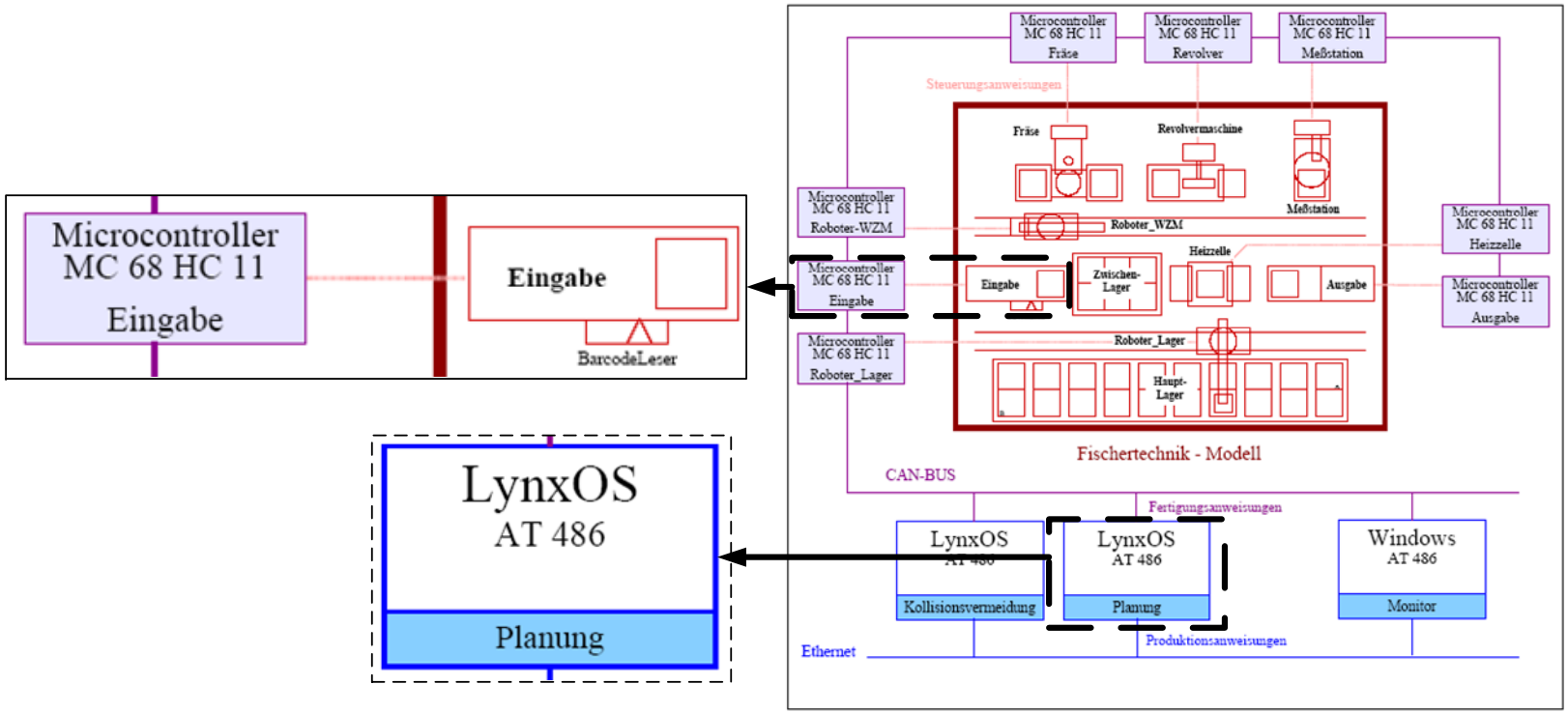
Synchrone Unterbrechungen (Traps/Exceptions)

- Werden durch das Programm selbst ausgelöst, d.h. dasselbe Programm, ausgeführt mit denselben Parametern wird in der Regel an derselben Stelle dieselbe Unterbrechung auslösen (vorhersagbar in dieselbe „Falle“ laufen)
- **Auslösung bei Fehler** – Ausnahme/Exception, Beispiele:
  - Arithmetikfehler (Division by zero, overflow, not-a-number NaN, …)
  - Speicherfehler (Page Fault, segment Fault, memory full, …)
  - Befehlsfehler (Illegal instruction, privileged instruction, bus error, …)
  - Peripheriefehler (End-of-file EOF, channel blocked, unknown device, …)
- Bei Exceptions **nur dann** Rückkehr an den Auftrittspunkt, wenn die Fehlerbedingung im Ausnahmebehandler beseitigt werden kann, andernfalls Abbruch (resumption vs. termination)
- **Auslösung durch spezifische Instruktion**: Breakpoint, SWI, TRAP, INT, … entweder zum Zwecke des „Debuggings“ oder zum Aufruf von Betriebssystem-Diensten (z.B. MS-DOS „INT 21h“, siehe z.B. http://en.wikipedia.org/wiki/MS-DOS_API)
- Traps können auch benutzt werden, um einen Hardware-Interrupthandler zu testen.

Asynchrone Unterbrechungen (Interrupts)

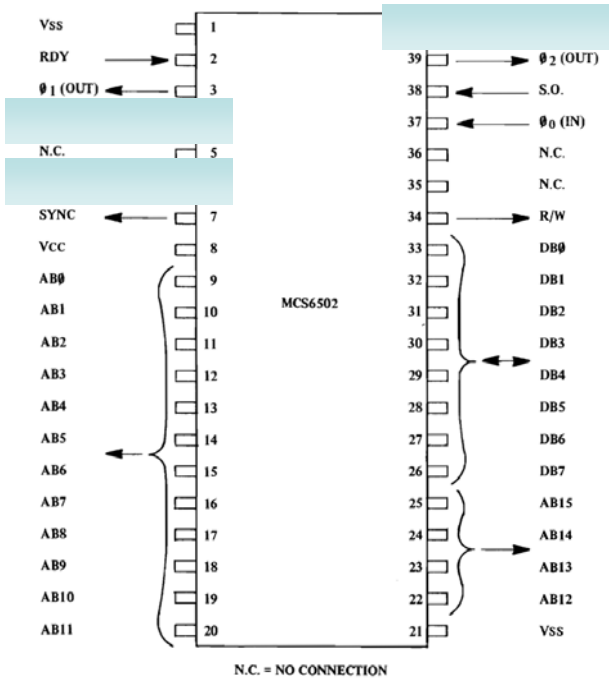
- Werden durch externe Prozesse ausgelöst, d.h. sind bezüglich des genauen Auftrittszeitpunkts unvorhersagbar und zumeist nicht reproduzierbar
- Beispiele:
  - Signalisierung „normaler“ externer Ereignisse durch periphere Einheiten (Timer, Schalter, Grenzwertüber-/unterschreitung, …)
  - Warnsignale der Hardware (Energiemangel, „Watchdog-timer“ abgelaufen, …)
  - Beendigung einer Ein-/Ausgabeoperation (Wort von serieller Schnittstelle komplett empfangen oder komplett gesendet, Operation von Coprozessor (DMA, FPU) komplett, …)
- Die Unterbrechungsbehandlung muß *nebeneffektfrei* verlaufen, d.h. das Hauptprogramm darf nach Abschluß der Behandlung keinen veränderten Ausführungs-Kontext vorfinden.
- Aber: typischerweise wird der Behandler zum Zwecke der Kommunikation über globale Variable mit dem Hauptprogramm kommunizieren.

Beispiel: Modellfabrik (Praktikum) mit Prozessoren und INT-Quellen



Technische Realisierung von Interrupts

- Zur Realisierung besitzen Prozessoren einen oder mehrere spezielle Interrupt-Eingänge (typ. IRQ oder INT-Anschluß). Wird ein Interrupt aktiviert, so führt dies zur Ausführung einer Unterbrechungs-behandlungsroutine.
- Das Auslösen der Unterbrechungsroutine entspricht einem Unterprogrammaufruf. Der Programmablauf wird an einer anderen Stelle fortgesetzt und nach Beendigung der Routine (normalerweise) an der unterbrochenen Stelle fortgefahren.



Pin-Belegung des MOS 6502-Prozessors (z.B. Commodore 64). Quelle: MCS 6500 HW-Manual, MOS Technology, Jan. 1976

Durchführung einer einfachen INT-Behandlung

1. IRQ-Anschluß wird durch peripheres Gerät aktiviert
2. Wenn Interrupts momentan zugelassen sind: Beendigung der Abarbeitung der gerade noch laufenden Instruktion
3. Sicherung der Register des Prozessors (Prozessorkontext) auf dem Stapelspeicher (Stack) – insbesondere Sicherung des Programmzählers; dafür spezielle Instruktionen verfügbar
4. Sprung an den Behandler (entspricht Laden des Programmzählers mit der Programmstartadresse des Behandlers) – dies kann auf verschiedene Arten erfolgen, siehe unten
5. Ausführung des Codes des Behandlers, an dessen Ende steht ein „Return from Interrupt“-Befehl; dabei Signalisierung an Peripherie, daß Behandlung abgeschlossen
6. Zurückladen des gesicherten Prozessorregistersatzes, Rücksprung (= Laden des Programmzählers mit der Adresse der Instruktion, die auf diejenige folgt, an der die Unterbrechung auftrat).

Beispiel INT beim 6502

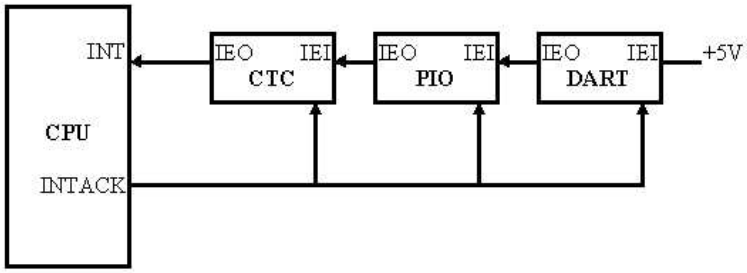
- Bei Vorliegen eines IRQ wird der 16-Bit Programmzähler mit dem Inhalt der 8-Bit Adressen FFFE und FFFF geladen.
- FFFE/F enthält Adresse des (= *Vektor* auf) IRQ-Behandler
- FFFC/D enthält Adresse des Reset-Behandlers
- FFFA/B enthält Adresse des NMI-Behandlers. NMI: Non-Maskable-Interrupt, kann nicht abgeschaltet werden (also auch nicht durch fehlerhaftes Programm)

Speicher-Aufteilung für 6502 (in Hex-Adressen)

0000-00FF - RAM for Zero-Page & Indirect-Memory Addressing
0100-01FF - RAM for Stack Space & Absolute Addressing
0200-3FFF - RAM for programmer use
4000-7FFF - Memory mapped I/O
8000-FFF9 - ROM for programmer usage
FFFA - Vector address for NMI (low byte)
FFFB - Vector address for NMI (high byte)
FFFC - Vector address for RESET (low byte)
FFFD - Vector address for RESET (high byte)
FFFE - Vector address for IRQ & BRK (low byte)
FFFF - Vector address for IRQ & BRK (high byte)

Behandlung mehrerer Quellen von Interrupts

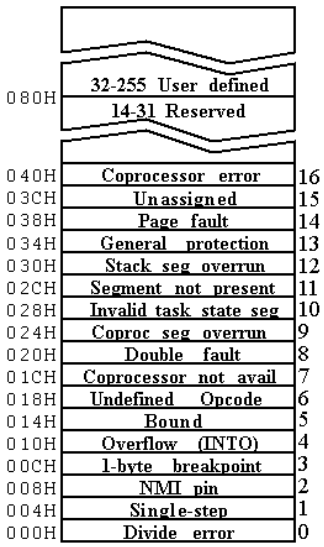
- Beim 6502 nur *ein* IRQ-Eingang und nur *ein* Vektor auf *einen* Behandler. Bei mehreren Quellen für Interrupts (z.B. serielle Schnittstelle, Parallele Schnittstelle, Timer) werden daher die Interrupt-Ausgänge dieser Einheiten „verodert“ und an den IRQ-Eingang gelegt. Alle Quellen sind damit gleichberechtigt.
- Damit ist keine automatische HW-Priorisierung (nach Wichtigkeit der eintreffenden Interrupts) möglich.
- Der Behandler muß nach dem Auftritt des Interrupts Quelle für Quelle abfragen, welche de Interrupt verursacht hat (implizite Priorisierung je nach Abfragereihenfolge)
- Einfache Möglichkeit der HW-Priorisierung: Daisy-Chain (Prioritätskette). Die Einheit, die am nächsten an der CPU liegt, hat die höchste Priorität (siehe Bild rechts) und sperrt Interrupts anderer Quellen solange aus, bis der eigene Interrupt abgearbeitet ist (INTACK durch Prozessor signalisiert).



Daisy-Chaining beim Z80-Prozessor
CTC: Counter-Timer-Circuit
PIO: Parallel In-/Out
DART: Dual Asynchronous Receiver Transmitter

Vektorisierte Interrupt-Behandlung

- Verfügt ein Rechnersystem über viele Interrupt-Quellen (wie typischerweise im Bereich eingebetteter Systeme), ist es zweckmäßig, für jedes Gerät (mindestens) einen Behandler vorzusehen und diesen nach dem Ereignisauftritt auch direkt ausführen zu können.
- Dazu Einführung von vektorbasierten Interrupt-Systemen.
- Prinzip: Gerät erzeugt nicht nur einen Interrupt, sondern liefert dem Prozessor parallel auch eine eigene Kennung (z.B. 8-Bit Wert A). Dieser Wert A verweist auf einen Tabelleneintrag, an dem sich die Einsprung-Adresse des Behandlers befindet.
- Exceptions und Interrupts folgen dem gleichen Schema, d.h. Tabelleneintrag = Einsprungadresse für Exception/Interrupt-Handler



The interrupt vector table is located in the first 1024 bytes of memory at addresses 000000H through 0003FFH.

There are 256 4-byte entries (segment and offset in real mode).

Seg high	Seg low	Offset high	Offset low
Byte 3	Byte 2	Byte 1	Byte 0

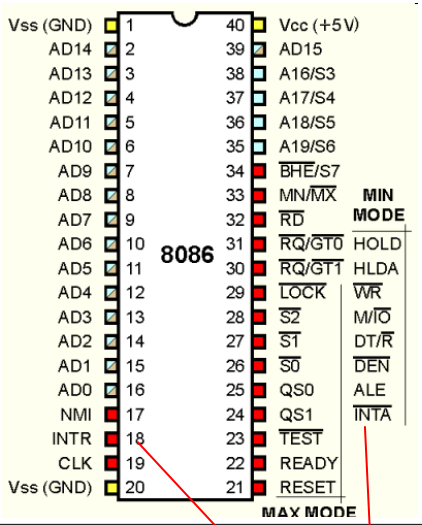
- Typische Interrupt-Vektor-Tabelle eines 8086-Systems im „Real Mode“
- 32-Bit-Adresse (4 Byte) pro Eintrag
- 0 bis 31 sind Prozessor-interne Ausnahmen
- Hardware-Interrupts können über Interrupt-Controller auf beliebige Tabellenplätze gelegt werden

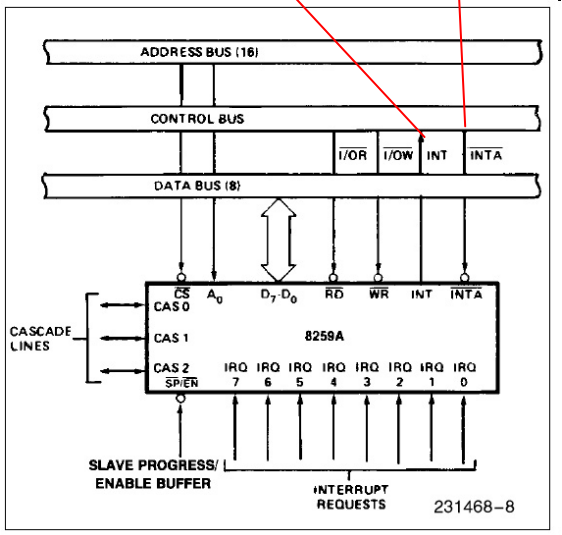
Interrupt-(Priority)-Controller

- Typischerweise wird mit der vektorbasierten Verwaltung auch eine Prioritätsverwaltung eingeführt. Klassisches Beispiel: der Interrupt-Controller 8259A des IBM-PC.
- Verwaltet 8 Hardware-Interrupts und kann diese mit Prioritäten belegen. Ist kaskadierbar, der PC/AT hatte zwei 8259A.
- Typen der Interrupts beim PC/AT:

00	Systemtaktgeber	08	Echtzeitsystemuhr
01	Tastatur	09	Frei
02	Programmierbarer Interrupt-Controller	10	Frei
03	Serielle Schnittstelle COM2 (E/A-Bereich 02F8)	11	Frei
04	Serielle Schnittstelle COM1 (E/A-Bereich 03F8)	12	PS/2-Mausanschluss
05	Frei, oft Soundkarte oder LPT2	13	Koprozessor (ob separat oder in CPU integriert)
06	Diskettenlaufwerk	14	Primärer IDE-Kanal
07	Parallel (Drucker-) Schnittstelle LPT1 (E/A-Bereich 0378)	15	Sekundärer IDE-Kanal

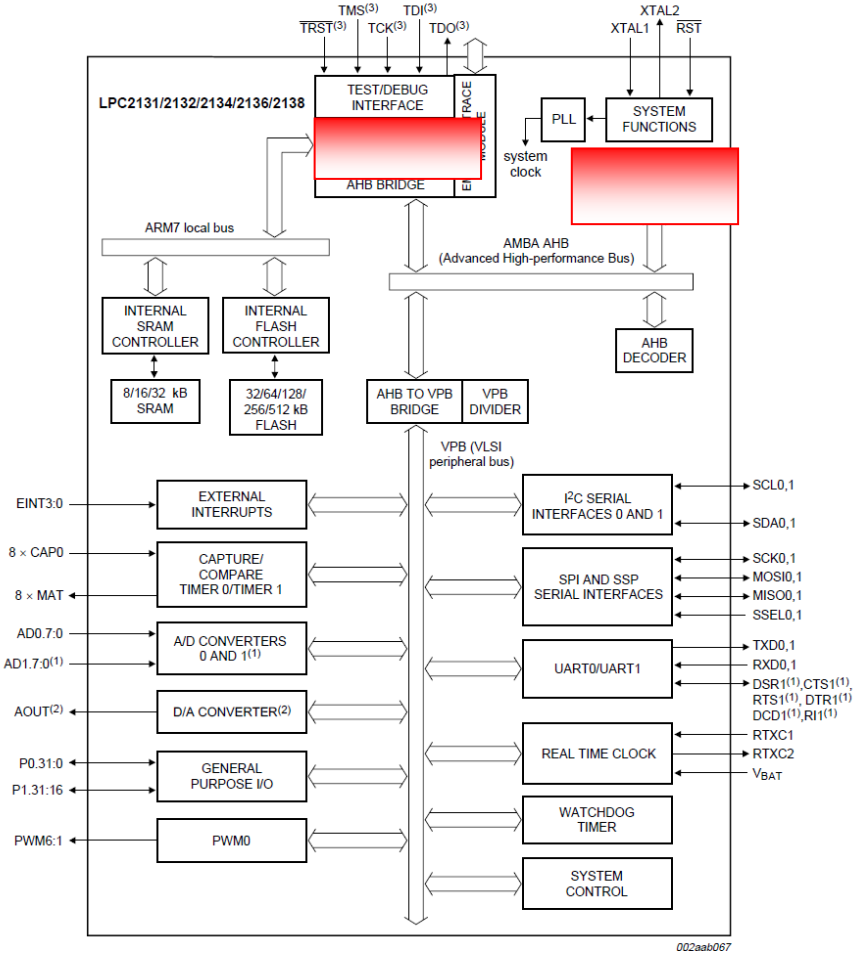
- Heute Standard (seit ca. 2002): Nachfolgekonzept, Intel APCI 82093AA, sehr viel höhere Anzahl von Interrupts realisiert





Interrupts an einem Micro-Controller

- Häufig eingesetztes System der 32-Bit-Klasse: ARM (Acorn Risc Machine)
- Hier betrachtet LPC 2138 von NXP, 32-Bit ein-Chip Controller mit Echtzeit-Uhr (RTC), UART, I2C-Bus, USB, A/D, D/A, etc.
- Alle peripheren einheiten auf dem Chip können Interrupts auslösen
- Alle Interrupts sind priorisierbar
- Dazu ist ein VIC (Vectored Interrupt Controller) vorgesehen mit 32 Interrupts, davon 16 vektorisiert



Acorn founders
Hermann Hauser and Chris Curry
(http://atterer.net/acorn.html)

LPC-Interrupt-Controller VIC: Register

Table 33: VIC register map

Name	Description	Access	Reset value[1]	Address
VICIRQStatus	IRQ Status Register. This register reads out the state of those interrupt requests that are enabled and classified as IRQ.	RO	0	0xFFFF F000
VICFIQStatus	FIQ Status Requests. This register reads out the state of those interrupt requests that are enabled and classified as FIQ.	RO	0	0xFFFF F004
VICRawIntr	Raw Interrupt Status Register. This register reads out the state of the 32 interrupt requests / software interrupts, regardless of enabling or classification.	RO	0	0xFFFF F008
VICIntSelect	Interrupt Select Register. This register classifies each of the 32 interrupt requests as contributing to FIQ or IRQ.	R/W	0	0xFFFF F00C
VICIntEnable	Interrupt Enable Register. This register controls which of the 32 interrupt requests and software interrupts are enabled to contribute to FIQ or IRQ.	R/W	0	0xFFFF F010
VICIntEnClr	Interrupt Enable Clear Register. This register allows software to clear one or more bits in the Interrupt Enable register.	WO	0	0xFFFF F014
VICSoftInt	Software Interrupt Register. The contents of this register are ORed with the 32 interrupt requests from various peripheral functions.	R/W	0	0xFFFF F018
VICSoftIntClear	Software Interrupt Clear Register. This register allows software to clear one or more bits in the Software Interrupt register.	WO	0	0xFFFF F01C
VICProtection	Protection enable register. This register allows limiting access to the VIC registers by software running in privileged mode.	R/W	0	0xFFFF F020
VICVectAddr	Vector Address Register. When an IRQ interrupt occurs, the IRQ service routine can read this register and jump to the value read.	R/W	0	0xFFFF F030
VICDefVectAddr	Default Vector Address Register. This register holds the address of the Interrupt Service routine (ISR) for non-vectored IRQs.	R/W	0	0xFFFF F034
VICVectAddr0	Vector address 0 register. Vector Address Registers 0-15 hold the addresses of the Interrupt Service routines (ISRs) for the 16 vectored IRQ slots.	R/W	0	0xFFFF F100
VICVectAddr1	Vector address 1 register.	R/W	0	0xFFFF F104
VICVectAddr2	Vector address 2 register.	R/W	0	0xFFFF F108
VICVectAddr3	Vector address 3 register.	R/W	0	0xFFFF F10C
VICVectAddr4	Vector address 4 register.	R/W	0	0xFFFF F110
VICVectAddr5	Vector address 5 register.	R/W	0	0xFFFF F114
VICVectAddr6	Vector address 6 register.	R/W	0	0xFFFF F118
VICVectAddr7	Vector address 7 register.	R/W	0	0xFFFF F11C
VICVectAddr8	Vector address 8 register.	R/W	0	0xFFFF F120
VICVectAddr9	Vector address 9 register.	R/W	0	0xFFFF F124
VICVectAddr10	Vector address 10 register.	R/W	0	0xFFFF F128
VICVectAddr11	Vector address 11 register.	R/W	0	0xFFFF F12C

Table 33: VIC register map

Name	Description	Access	Reset value[1]	Address
VICVectAddr12	Vector address 12 register.	R/W	0	0xFFFF F130
VICVectAddr13	Vector address 13 register.	R/W	0	0xFFFF F134
VICVectAddr14	Vector address 14 register.	R/W	0	0xFFFF F138
VICVectAddr15	Vector address 15 register.	R/W	0	0xFFFF F13C
VICVectCntl0	Vector control 0 register. Vector Control Registers 0-15 each control one of the 16 vectored IRQ slots. Slot 0 has the highest priority and slot 15 the lowest.	R/W	0	0xFFFF F200
VICVectCntl1	Vector control 1 register.	R/W	0	0xFFFF F204
VICVectCntl2	Vector control 2 register.	R/W	0	0xFFFF F208
VICVectCntl3	Vector control 3 register.	R/W	0	0xFFFF F20C
VICVectCntl4	Vector control 4 register.	R/W	0	0xFFFF F210
VICVectCntl5	Vector control 5 register.	R/W	0	0xFFFF F214
VICVectCntl6	Vector control 6 register.	R/W	0	0xFFFF F218
VICVectCntl7	Vector control 7 register.	R/W	0	0xFFFF F21C
VICVectCntl8	Vector control 8 register.	R/W	0	0xFFFF F220
VICVectCntl9	Vector control 9 register.	R/W	0	0xFFFF F224
VICVectCntl10	Vector control 10 register.	R/W	0	0xFFFF F228
VICVectCntl11	Vector control 11 register.	R/W	0	0xFFFF F22C
VICVectCntl12	Vector control 12 register.	R/W	0	0xFFFF F230
VICVectCntl13	Vector control 13 register.	R/W	0	0xFFFF F234
VICVectCntl14	Vector control 14 register.	R/W	0	0xFFFF F238
VICVectCntl15	Vector control 15 register.	R/W	0	0xFFFF F23C

LPC-2138-Interrupt-Quellen

5.4.3 Raw Interrupt status register (VICRawIntr - 0xFFFF F008)

This is a read only register. This register reads out the state of the 32 interrupt requests and software interrupts, regardless of enabling or classification.

Table 38: Raw Interrupt status register (VICRawIntr - address 0xFFFF F008) bit allocation
Reset value: 0x0000 0000

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	-	-	-	-
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	-	-	AD1	BOD	I2C1	AD0	EINT3	EINT2
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	EINT1	EINT0	RTC	PLL	SPI1/SSP	SPI0	I2C0	PWM0
Access	RO	RO	RO	RO	RO	RO	RO	RO
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT
Access	RO	RO	RO	RO	RO	RO	RO	RO

LPC-2138-Interrupt-Programmierung

- Beispiel:
  Regelmäßiges Auslösen eines Interrupts durch Timer 0 (z.B. alle 10ms) und Hochzählen einer Variable num_calls
- **1.Schritt:** Definition des Behandlers (ISR)

```
int volatile num_calls;

void IRQ_Timer0(void)  __attribute__((naked)); // Erklärung naked siehe nächste Folie

void IRQ_Timer0 (void)
{
        ISR_ENTRY();            // Eingangssequenz-Makro zur Sicherung des Prozessor-Status
        num_calls++;            //  Zähle die Anzahl der Aufrufe
        T0IR = 0x01;            //  Clear Interrupt Flag
        VICVectAddr = 0x00;     // Update priority hardware
        ISR_EXIT();             // Abschluß-Makro zur Wiedergherstellung des Prozessor-Status
}
```

- Alternativ kann in gcc auch ohne die Makros verfahren werden, wenn Prozedurkopf lautet:
  `void __attribute__ ((interrupt)) IRQ_Timer0 (void)`
  aber nicht für jeden Prozessor verfügbar!

LPC-2138-Interrupt-Programmierung

- void IRQ_Timer0(void) __attribute__((naked));

  Gcc-Manual (http://gcc.gnu.org/onlinedocs/gcc-4.4.2/gcc.pdf) Seite 289:

  ((naked))

  Use this attribute on the ARM, AVR, IP2K and SPU ports to indicate that the specified function does not need prologue/epilogue sequences generated by the compiler. It is up to the programmer to provide these sequences. The only statements that can be safely included in naked functions are asm statements that do not have operands. All other statements, including declarations of local variables, if statements, and so forth, should be avoided. Naked functions should be used to implement the body of an assembly function, while allowing the compiler to construct the requisite function declaration for the assembler.

- Die Macros ISR_ENTRY() und ISR_EXIT() sind auf der Webseite verfügbar (armVIC.h).

LPC-2138-Interrupt-Programmierung

- **2.Schritt:** Programmierung des VIC

```
void sysInit (void)
{
    VICIntEnClear = 0xFFFFFFFF;          // clear all interrupts
    VICIntSelect = 0x00000000;           // clear all FIQ selections
    VICDefVectAddr = (uint32_t)reset;    // point unvectored IRQs to reset()
    VICVectAddr2 = (uint32_t) IRQ_Timer0;  // Entry-Point IRQ-Timer0 (= Address of ISR); Prio=2
    VICVectCntl2 = 0x20 | 0x04;          // For IRQ slot 2 set „enable slot“ Bit 5 AND activate for
                                         // Timer0 interrupts
    VICIntEnable = 1 << 0x04;            // Enable Tomier 0 Interrupts
}
```

LPC-2138-Interrupt-Programmierung

- **3.Schritt:** Programmierung des Timers 0

```
void timerInit (void)
{
    T0TC = 0;           // Timer-Value (to be incremented by clock)
    T0PR = 0;           // Prescaler = 0 (Vorteiler für Timer-Eingangstakt)
    T0MR0 = 240000;     // Match-Register. Generate Interrupt when Timer has reached 240000
    T0MCR = 0x03;       //  Generate interrupt on match and reset to zero
    T0TCR = 0x01;       // Timer Control Register: enable counting
    T0IR = 0x01;        // Enable Timer 0 Interrupts
}
```