Inhaltsübersicht

Kapitelübersicht

I. Was sind Exceptions und Interrupts?
II. Interrupts im Cortex-M0
III. Interrupt Prioritäten

Was sind Exceptions und Interrupts?

- Exceptions (dt. Ausnahmen) sind Ereignisse, die den normalen Programmablauf unterbrechen und eine dem Ereignis zugeordnete Funktion ausführen (Exception Handler).
- Als Interrupts werden Exceptions bezeichnet, die von außen oder durch Peripherieeinheiten ausgelöst werden.
  - Neben diesen Interrupts gibt es auch Ereignisse im Prozessorsystem selbst, die zu einer Exception führen können (z.B. falscher Buszugriff)

Polling vs. Interrupts

- Häufiges Problem bei Mikrocontrollern:
  - „Abfrage“ von Peripherieeinheiten, z.B.
    - Überlauf des Timers
    - Senden/Empfangen von Bytes im UART
    - A/D-Wandler
- Zwei Möglichkeiten:
  - Zyklische Abfrage („Polling“) der Peripherie durch den Prozessor
    - Prozessor ist mit Abfragen beschäftigt, verbraucht daher viel Prozessorleistung
    - Wenn nicht schnell genug abgefragt wird, kann ein Ereignis verpasst werden.
  - Unterbrechung („Interrupt“) des Prozessors durch die Peripherie
    - Prozessor kann andere Aufgaben abarbeiten
    - Einfache Möglichkeit der Priorisierung

Funktionsweise von Interrupts

- Ein **Interruptereignis** einer Peripherieeinheit (Interruptquelle) führt dazu, dass die CPU das laufende Programm (Vordergrund, base level) unterbricht und eine dem Ereignis zugeordnete **Interrupt Service Routine** (auch als „Interrupt Handler“ bezeichnet) aufruft (Hintergrund, interrupt-level). Dies entspricht dem Aufruf eines Unterprogramms.
- Am Ende der ISR kehrt die CPU wieder zum Hauptprogramm zurück (Return-from-Interrupt).

Funktionsweise von Interrupts (2)

- Der Unterschied zu einem Unterprogrammaufruf:
  - Der Interrupt Handler wird nicht aus dem Hauptprogramm (Software) aufgerufen, sondern durch die Hardware
  - Es ist a priori nicht bekannt, wann der Interrupt auftritt. Das Interrupt-Ereignis ist „asynchron“ zum Programmablauf (z.B. Tastatur), ein UP-Aufruf ist synchron.
- Interrupt Organisation:
  - Ein-/Ausschalten von Interrupts
  - Priorisierung von Interrupts
  - Verarbeitung von Interrupts, Flags

Programmablauf mit Interrupts

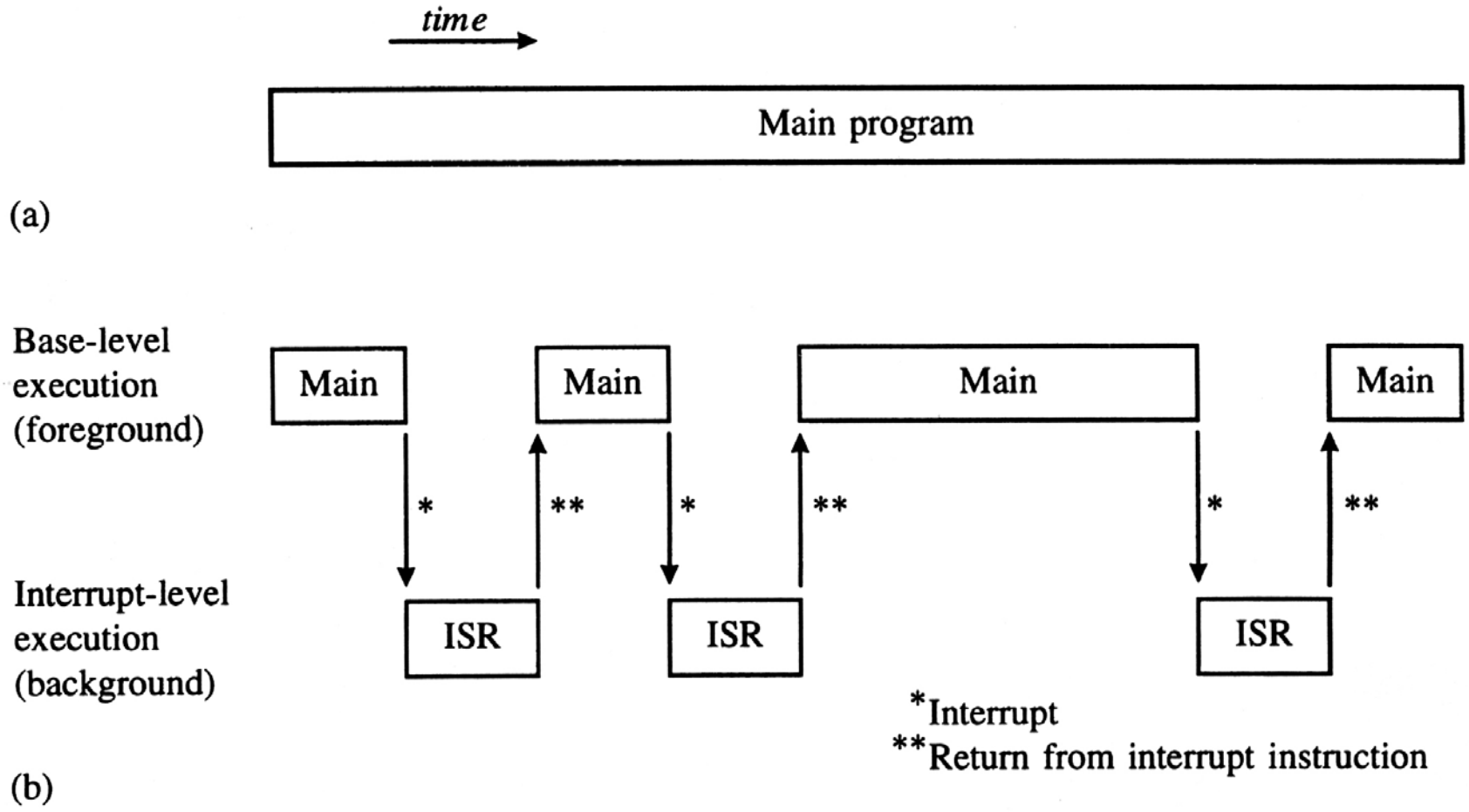


FIGURE 6–1
Program execution with and without interrupts (a) Without interrupts (b) With interrupts

Kapitelübersicht

I. Was sind Exceptions und Interrupts?

II. Interrupts im Cortex-M0

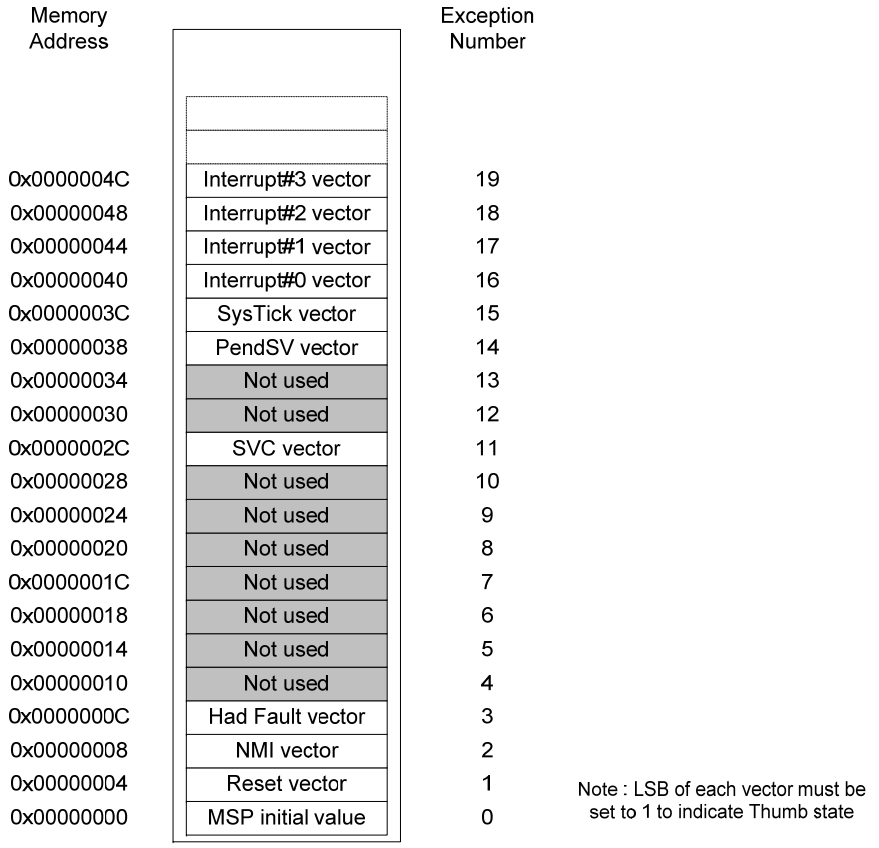
III. Interrupt Prioritäten

Interrupts im Cortex-M0

- Die Interrupts werden im Cortex-M0 durch den NVIC (Nested Vectored Interrupt Controller) verwaltet
- Der NVIC unterstützt neben den Exceptions bis zu 32 Interrupts von Peripherieeinheiten
  - Die Exceptions haben die Exception-Nummern 1-15
  - Die Interrupts haben die Exception-Nummern 16-47
- Damit ein Interrupt wirksam wird, muss er freigeschaltet werden („enable“).
- Kann ein Interrupt nicht sofort durch die Ausführung des Handlers behandelt werden, so wird gespeichert, dass der Interrupt aufgetreten ist („pending“) und der Handler zu einem späteren Zeitpunkt ausgeführt.

Exception-Vektor-Tabelle

- Die Anfangsadresse des jeweiligen Exception-Handlers wird in der so genannten „Vektor-Tabelle“ gespeichert.
- Diese Informationen werden vom Linker ermittelt und in das Image eingetragen.

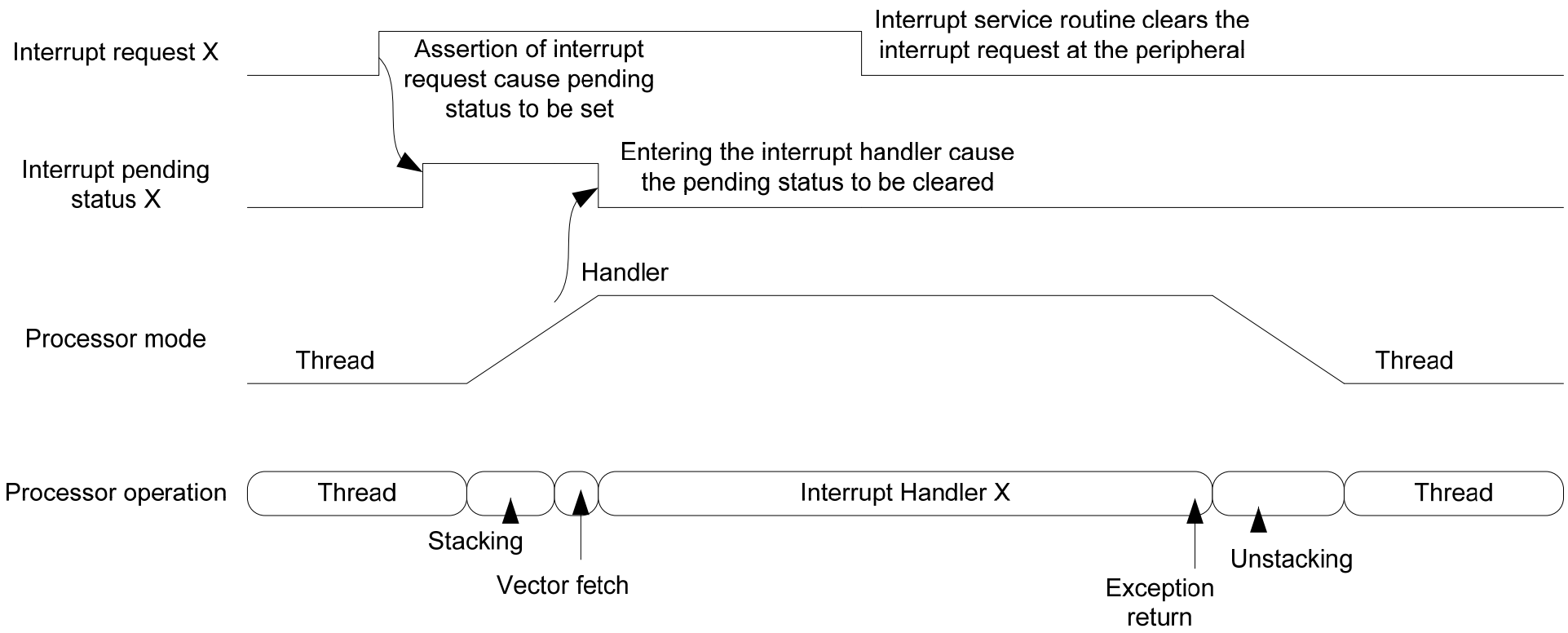


Bildquelle: Joseph Yiu, The Definitive Guide to the ARM Cortex-M0

Wie wird der Interrupt behandelt?

- Das Auftreten eines Interruptereignisses erzeugt einen „Interrupt-Request“ der Peripherieeinheit.
- Dies führt dazu, dass der Interrupt als „Pending“ im NVIC gespeichert wird.
- Durch das „Stacking“ (und „Unstacking“) wird der Prozessor-Zustand für das Hauptprogramm gerettet.
- Der Cortex holt die Anfangsadresse des Handlers aus der „Vektortabelle“ und führt den Handler aus.
- Beim Eintritt in den Handler wird der „Pending“-Status im NVIC rückgesetzt.
- Im Handler kann nun (nach Bedarf) auch der Interrupt-“Request“ gelöscht werden

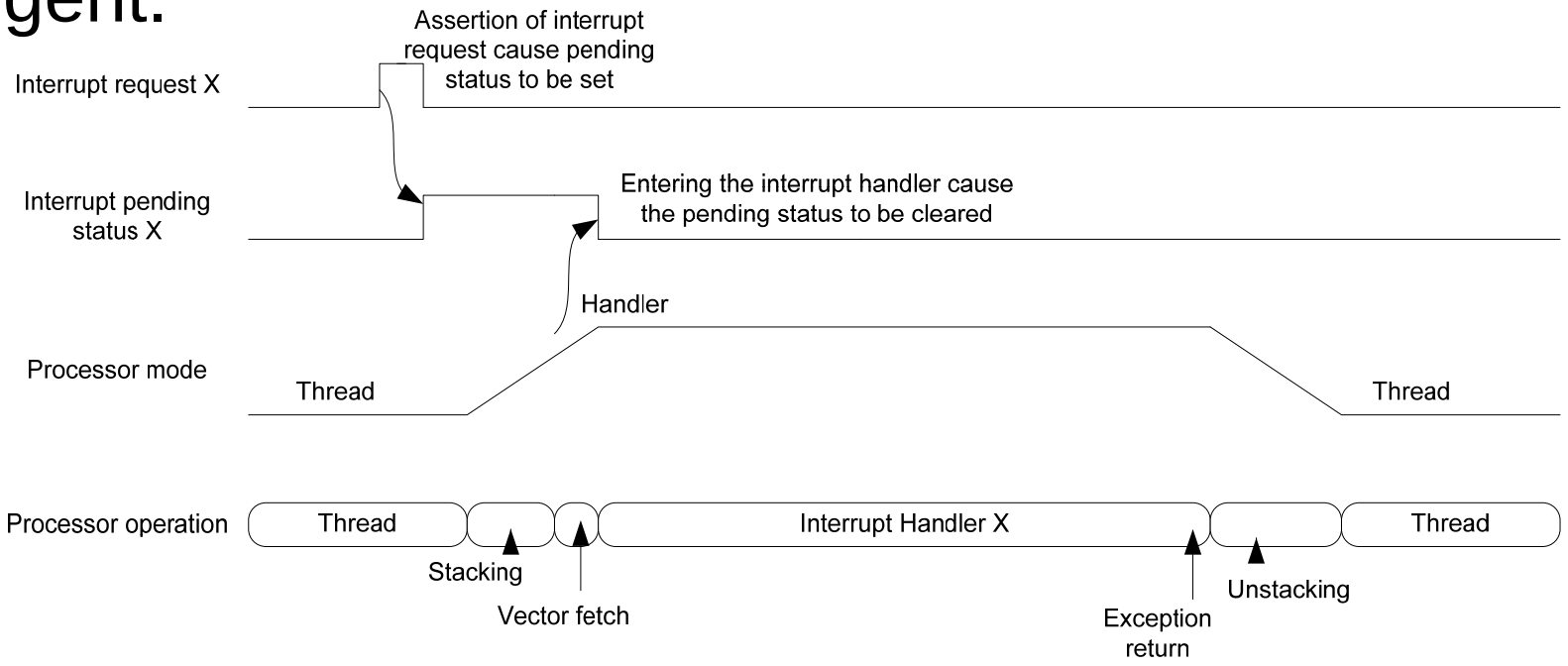
Ausführen des Interrupts im Cortex



Bildquelle: Joseph Yiu, The Definitive Guide to the ARM Cortex-M0

Interrupt-Request als Puls

- Manche Peripherieeinheiten erzeugen nur einen Puls beim Interruptereignis. Der Pending-Status stellt hier sicher, dass der Interrupt nicht verloren geht.



Bildquelle: Joseph Yiu, The Definitive Guide to the ARM Cortex-M0

Der „Hauptschalter“ für Interrupts

- Der Cortex verfügt über ein PRIMASK-Register (siehe Kapitel 2)
- Dieses besteht nur aus einem einzigen Bit.
- Ist das Bit = 0, dann werden Interrupts und Exceptions vom Cortex akzeptiert (Default nach Reset)
- Ist das Bit = 1, dann werden nur einige wenige Exceptions akzeptiert.
- Damit können alle Interrupts z.B. für eine bestimmte Zeit deaktiviert werden.

Freischalten einzelner Interrupts

- Damit Interrupts wirksam werden, also der Handler ausgeführt wird, müssen zwei Dinge getan werden:
  - Die Peripherieeinheit muss so programmiert werden, dass sie „Interrupt Requests“ erzeugt.
  - Im NVIC muss der Interrupt freigeschaltet werden. Dies kann z.B. durch die Funktion **`NVIC_EnableIRQ(IRQn)`** vorgenommen werden (IRQn siehe nächste Folie).
- Das Sperren eines Interrupts erfolgt mit: **`NVIC_DisableIRQ(IRQn)`**

Einige Interrupt-Quellen im NUC130

- SysTick-Timer (eigentlich eine Exception) (**`SysTick_IRQn`**)
- Timer0 – Timer3 (**`TMR0_IRQn – TMR3_IRQn`**)
- UART0, UART1 (**`UART0_IRQn, UART1_IRQn`**)
- SPI0 – SPI3 (**`SPI0_IRQn – SPI3_IRQn`**)
- AD-Wandler (**`ADC_IRQn`**)

Wie schreibt man einen Interrupt Handler?

- Ein Handler kann als normale C-Funktion geschrieben werden.
- Er hat keine Argumente und keinen Return-Wert
- Der Name der Handler-Funktion ist in der Datei „startup_NUC1xx.s“ vorgegeben.
- Für den Handler für den SysTick-Timer muss z.B. der Name „SysTick_Handler“ benutzt werden.

Beispiel: SysTick als Zeitgeber

```
#include "BoardConfig.h"
#include "Driver_M_Dongle.h"
#include "init.h"

//Global variable
uint8_t counter = 0;

//Main called by startup_NUC1xx.s
int main (void){

   DrvSystem_ClkInit();   //Setup clk system
   Board_Init();          //Initialize peripherals
   //Configure SysTick: Arg is the number of clocks (@12 MHz)
   //until the timer generates an interrupt, enables SysTick interrupts
   //for 1 sec periodical interrupt we need 12000000 clocks
   //Priority is set to 3
   SysTick_Config(12000000);

   while(1) {
      GPIOE->DOUT_BYTE1 = ~counter;
   }
}

//SysTick Interrupt Handler
void SysTick_Handler(void){
   counter++;
}
```

Funktionsweise des Beispiels

- Die Funktion „SysTick_Config()“ schaltet den Interrupt des SysTick frei und stellt die Anzahl der Takte ein, die zu zählen sind, bis der Interrupt ausgelöst wird (Ereignis).
- Bei Eintreten des Interruptereignisses wird die Handler-Funktion „SysTick_Handler()“ ausgeführt.
- Diese inkrementiert eine globale Variable, welche im Hauptprogramm auf der LED-Zeile ausgegeben wird.

Kapitelübersicht

I. Was sind Exceptions und Interrupts?

II. Interrupts im Cortex-M0

III. Interrupt Prioritäten

Interrupt Prioritäten

- Sind mehrere Interruptquellen gleichzeitig aktiv, so muss entschieden werden, welcher Handler als erstes ausgeführt wird.
- Wird gerade ein Handler ausgeführt, so kann möglicherweise ein weiterer Interrupt einer anderen Quelle auftreten, der sofort behandelt werden muss. Der laufende Handler muss unterbrochen werden.
- Diese Probleme werden durch programmierbare Prioritäten im NVIC gelöst.

Programmierung von Prioritäten

- Jeder Interrupt verfügt über ein 8-Bit-Prioritätsregister, mit welchem vier Prioritätsstufen programmiert werden können.
  - Der kleinere Wert hat die höhere Priorität
  - 0x00 (=0, höchste), 0x40 (=1), 0x80 (=2), 0xC0 (=3, niedrigste)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Implemented		Not implemented, read as zero					

- Prozessor-Exceptions haben feste negative Werte und damit noch höhere Priorität als Interrupts

Bildquelle: Joseph Yiu, The Definitive Guide to the ARM Cortex-M0

Programmierung mit CMSIS-Funktionen

- Die Priorität eines Interrupts kann am einfachsten mit folgender Funktion eingestellt werden:
  `NVIC_SetPriority(IRQn, priority)`
- **`IRQn`** ist dabei die Interruptquelle (siehe Folie 16) und **`priority`** ist ein Wert zwischen 0 und 3, dieser wird entsprechend ausgerichtet in das Prioritätsregister geschrieben.
- Mit der Funktion
  `NVIC_GetPriority(IRQn)`
  kann die Priorität eines Interrupts abgefragt werden.

Wann akzeptiert der Prozessor einen Interrupt?

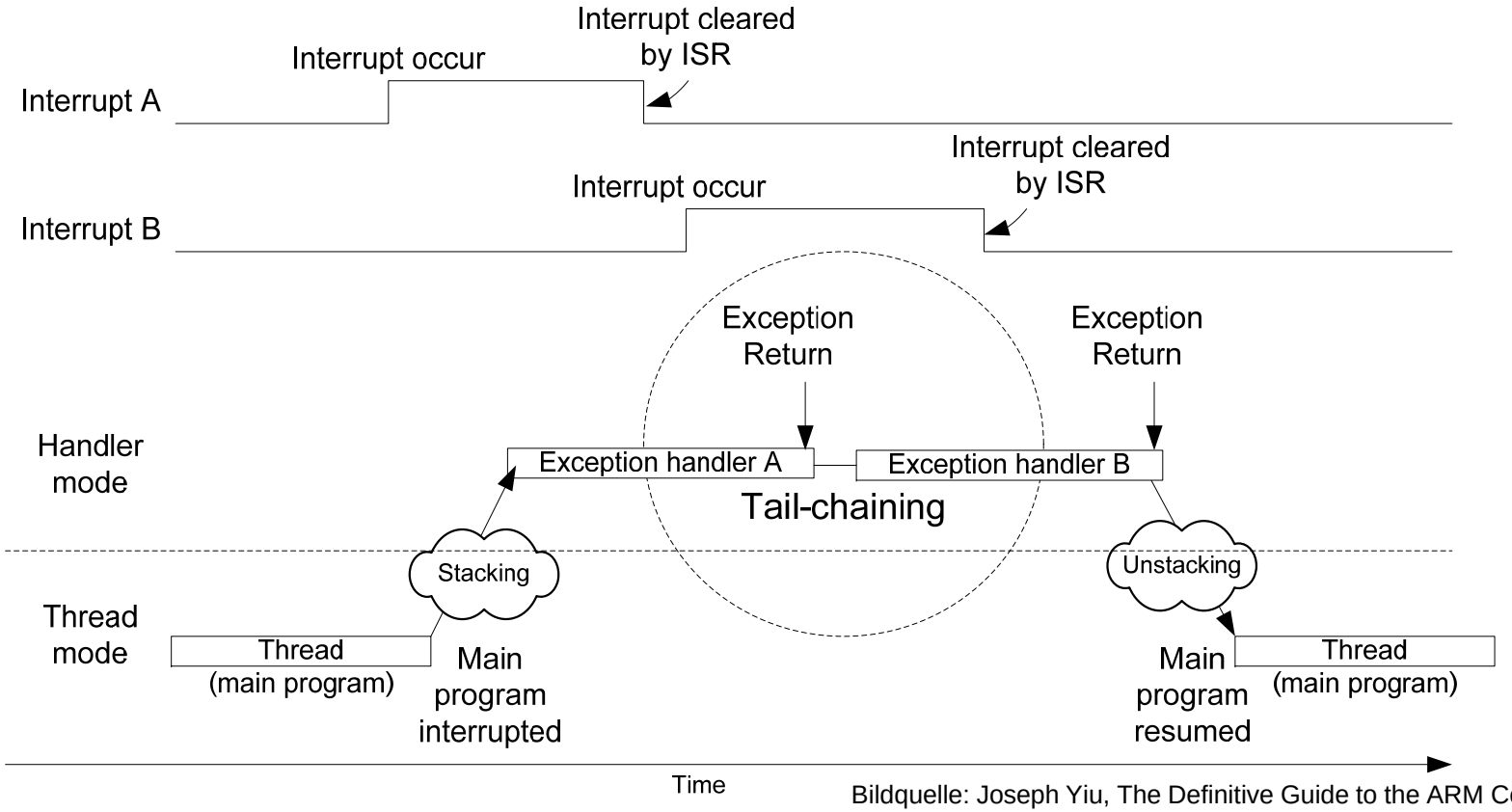
- Damit der Prozessor einen Interrupt akzeptiert und damit der Handler ausgeführt werden kann, müssen folgende Voraussetzungen erfüllt sein:
  - Der Interrupt muss im NVIC freigeschaltet sein
  - Es läuft gerade kein Handler der gleichen oder einer höheren Priorität
- Die Ausführung des Handlers unterbricht den normalen Programmablauf, nach Ausführung des Handlers wird dieser fortgesetzt

Geschachtelte Interrupts

- Tritt während der Ausführung eines Handlers ein weiterer Interrupt mit höherer Priorität auf, so wird der laufende Handler unterbrochen und der Handler des höher priorisierten Interrupts ausgeführt („nested interrupt“).
- Nach Ausführung des höher priorisierten Interrupt-Handlers wird der Handler des niedriger priorisierten Interrupts weiter ausgeführt.

„Tail-Chaining“

- Tritt während eines laufenden Handlers ein Interrupt der gleichen Priorität auf, so wird dessen Handler direkt im Anschluss ausgeführt („tail chaining“).



Bildquelle: Joseph Yiu, The Definitive Guide to the ARM Cortex-M0