**Networked Embedded Systems**

**Practicum 4: Interrupts and DMA**

**Group number: 8**

|  |  |  |
| --- | --- | --- |
| **Name** | **Student ID** | **Email** |
| Bernberger Sarah | k12112018 | sarahbernberger.sb@gmail.com |
| Grundner Simon | k12136610 | simon.grundner@gmail.com |

**18.06.2024**

Map of Content

[Theory Questions 3](#_Toc169657871)

[What is DMA? How does it work? 3](#_Toc169657872)

[What are the differences between Interrupt and DMA mode? 5](#_Toc169657873)

[Task A: Blink an LED with External Interrupts 6](#_Toc169657874)

[A.1. Implementation 6](#_Toc169657875)

[A.2. Discussion 6](#_Toc169657876)

[Task B: Blink an LED using Non-Blocking DMA and Timer Interrupts 7](#_Toc169657877)

[B.1. Implementation 7](#_Toc169657878)

[B.2. Results 7](#_Toc169657879)

[Task C: Read Magnetic Data from LIS3MDL Sensor in Non-Blocking Mode using Interrupts 8](#_Toc169657880)

[C.1. Implementation 8](#_Toc169657881)

[C.2. Results 8](#_Toc169657882)

[C.3. Discussion 8](#_Toc169657883)

[Task D: Read Magnetic Data from LIS3MDL Sensor in Non-Blocking Mode using DMA 9](#_Toc169657884)

[D.1. Implementation 9](#_Toc169657885)

[D.2. Discussion 9](#_Toc169657886)

[Index 10](#_Toc169657887)

[Figures 10](#_Toc169657888)

[Code Segments 10](#_Toc169657889)

[Tables 10](#_Toc169657890)

# Theory Questions

### What is DMA? How does it work?

DMA is short for “Direct Memory Access” and is a core component of a microcontroller to ensure constant data flow between a peripheral and the memory, meaning that data acquisition is not interrupted by CPU operations such as reading or processing the data. Data can be quickly moved by the DMA without any CPU action. This keeps the CPU resources free for other operations, which is especially important for time critical signals like audio real time sensor data.

#### Processing without a DMA-Controller

* CPU is constantly polling the peripheral
* Minimal processing time for other tasks
* Data provided by the peripheral during other processes is lost
* No consistent sampling period possible

Ein Bild, das Text, Screenshot, Schrift, Logo enthält.

Automatisch generierte Beschreibung

Figure 1: Interconnection

#### Processing with a DMA-Controller

* Stream samples directly into a queue
* Works independently from the CPU-Core
* Data loss only occurs when the FIFO-Buffer overflows

An application where the data transfer between memory and the peripheral is handled by a DMA-Controller would be setup as shown in Figure 2. Note that RX- and TX-Operation are handled by a different DMA-Stream with different queues.

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

Automatisch generierte Beschreibung

Figure 2: Simplified DMA Interconnection Diagram

#### Setting up DMA

A problem with the DMA controllers is that, although they can access almost all memory units, they cannot access the TCM[[1]](#footnote-2)-RAM, which is dedicated to the CPU. Therefore, a memory location accessible to both the CPU and the DMA controllers must be chosen. The most convenient memory unit for this purpose is AXI SRAM in D1. The connection is made through the Advanced Extensible Interface Matrix (AXI-M). This structure links the CPU to the Inter-Domain Buses (AHB), where the DMA controller and the Peripheral Buses (APB) are connected to as well.

The most important connections are highlighted in Figure 3. These connections are established by a bus multiplexer.

Ein Bild, das Text, Diagramm, parallel, Zahl enthält.

Automatisch generierte Beschreibung

Figure 3: Bus Interconnection Matrices

A simplified diagram shows how the CPU is connected to the internal SRAM.

*Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung*

Figure 4: Simplified AXI Interconnection description

The change of the memory unit requires changes made to the linker script to configure the memory location to be in the SRAM in D1. DTCMRAM must be replaced with RAM\_D1 for the following sections in *STM32H743XIHx\_FLASH.Id* file:

* .data
* .bss
* .\_user\_heap\_stack



Code Segment 1: .data field in linker script before change



Code Segment 2: .data field in linker script after change

Furthermore, the CPU contains two internal caches, I-Cache for loading instructions and D-Cache for data. The D-Cache can affect the functionality of DMA transfers, since it will hold the new data in the internal cache and do not write them to the SRAM memory. However, the DMA controller loads the data from SRAM memory and not D-Cache. Therefore, the D-Cache is globally disabled in Cube MX under *System Core > CORTEX\_M7 > Cortex Interface Settings*.

The *\_estack* variable in the linker script contains the address for the end of the stack. The default memory address, which is 0x20020000, is composed of the origin plus the length of the DTCMRAM. Since the TCM is no longer used, this field must be changed as well.

Update the section .user\_heap\_stack as follows:



Code Segment 3: .\_user\_heap\_stack in linker script

The end-of-stack variable is now set in this section to the dot-variable value at the shown location. The special linker dot-variable ‘.’ always contains the current output location counter.

### What are the differences between Interrupt and DMA mode?

#### Interrupt Mode

Setting up an interrupt eliminates the need to repeatedly check if a peripheral is ready. Instead, whenever the corresponding interrupt flag is set, a **callback function** is executed to handle an operation when the peripheral sends or receive data.

In contrast to Practicum 3, where the status register content of the sensor device was requested in each while-loop cycle, causing unnecessary traffic on the I2C bus, interrupts ensure that only necessary transmissions are made. This is particularly beneficial when multiple devices are connected to the bus.

With an interrupt-based implementation the CPU is signaled every time a byte is sent and must provide new data. This means, that other processes must wait until the transmission is executed.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Source | | | | Destination | | Type | Target |
| Domain | Bus | Peripheral | Signal | Signal | Peripheral |
| D2 | APB1 | I2C2 | i2c2\_wkup | WKUP23 | EXTI | D | CPU |

#### DMA Mode

The implementation with DMA looks similar to the interrupt driven method with the difference, that the DMA-Controller fully handles the data transfer. This makes the transmission a non-blocking operation and the CPU is only signaled if a transfer is complete instead of for every Byte sent. The CPU can simply provide data and send it onto a buffer without needing to wait until a word is transmitted. Individual DMA-Transfers do not involve the Processor and are generally faster.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Source | | | | Destination | | | |
| Domain | Bus | Peripheral | Signal | Signal | Peripheral | Bus | Domain |
| D2 | APB1 | I2C2 | i2c2\_rx\_dma | dmamux1\_req\_in35 | DMAMUX1 | AHB1 | D2 |
| i2c2\_tx\_dma | dmamux1\_req\_in36 |

# Task A: Blink an LED with External Interrupts

This task is similar to the task D of practicum 1 but now you are required to use external interrupts (EXTI) on PJ12 and PJ13 to detect user inputs. You should also use the timer for the blinking cycle (as introduced in practicum 2). The EXTI line (connected to PJ12) should be used to increase the cycle time while the EXTI line (connected to PJ13) should be used to decrease the cycle time. The LED located near the 7-Segment display should be used for digital output. The initial value of the blinking 2 cycle time should be 100 ms. The range of the blinking cycle time should be between 100 and 1000 ms. The respective interrupts either increase or decrease the current blinking cycle time by 100 ms. Pressing the buttons should update the blinking frequency with immediate effect.

**Requirements:**

* Minimum cycle time = **100 ms**
* Maximum cycle time = **1000 ms**
* Use **PJ7 (SEGDP)** as digital output
* Use **PJ12 (BTN1)** and **PJ13 (BTN2)** to generate external interrupts
* One step = **100 ms**

## A.1. Implementation

*Describe your solution. Add all relevant code snippets into a listing or as screenshots and describe their purpose. Remember to use generic methods with well-defined parameters!!*

## A.2. Discussion

*Describe your experiences (e.g., design decisions, problems, lesson learned). Which part of the code will be reusable?*

# Task B: Blink an LED using Non-Blocking DMA and Timer Interrupts

In this task, we will again work on our blinking LED. However, in this task, a user interface should be implemented. You are required to control the Timer 4 with user input via UART4. The user should be allowed to start and stop the timer. At each interrupt of Timer 4, the LED of the 7-segment display must be toggled.

For user input, the following command structure must be used: #X\*.

* ‘#’ stands for the start of a command
* ‘X’ stands for the command ID number:
  + 1 = Start timer
  + 2 = Stop timer
* ‘\*’ marks the end of a command

(Start command example: #1\*)

The UART4 interface must be configured for a non-blocking DMA receiving mode. The Timer 4 must be configured to generate an interrupt every 100 ms. Don’t forget to implement some error handling. Wrong commands should be handled gracefully. Give helpful feedback to the user in case of incorrect commands.

**Requirements:**

* Use **PJ7 (SEGDP)** for blinking
* Timer 4 generates periodic interrupts at **100ms**
* UART configured for non-blocking **DMA** receiving
* Use **#Command ID\*** as command message format
* Start Command ID = **1**
* Stop Command ID = **2**

## B.1. Implementation

*Describe your solution. Add all relevant code snippets into a listing or as screenshots and describe their purpose.**Remember to use generic methods with well-defined parameters!!*

## B.2. Results

*Add a screenshot of the UART output. Based on this screenshot, describe the functionality of your implementation.*

# Task C: Read Magnetic Data from LIS3MDL Sensor in Non-Blocking Mode using Interrupts

This task is an extension to task 1 of practicum 3. Previously, a polling method was implemented to read the X, Y, and Z-axis magnetic data from the LIS3MDL sensor. In this task, you are required to **implement interrupts** for I2C read and write. The respective addresses can be found in the datasheet and application note. After reading the magnetic data from the three axes, they must be displayed on a serial monitor of your choice. To do this, you are required to use UART4 of the STM32 MCU with the configuration specified in the tutorial named “**UART Interrupt and DMA Mode**”. The read values must be displayed in the following format:

**X: xxxx Gauss, Y: yyyy Gauss, Z: zzzz Gauss**

**Requirements:**

* Use the **I2C2 interface** to read data from the digital sensor
* Use the **UART4** interface to send data to the serial monitor
* Use the **interrupt mode** for reading/writing data
* Follow the display format mentioned above

## C.1. Implementation

*Describe your solution. Add all relevant code snippets into a listing or as screenshots and describe their purpose.**Remember to use generic methods with well-defined parameters!!*

## C.2. Results

*Add a screenshot of the UART output.*

## C.3. Discussion

*Describe your experiences (e.g., design decisions, problems, lesson learned). Which part of the code will be reusable?*

*What are the differences to your implementation in Practicum 3? Compare the implementations! What are the advantages and disadvantages?*

# Task D: Read Magnetic Data from LIS3MDL Sensor in Non-Blocking Mode using DMA

This task is a modification of the previous task. In this task, you need to use the **DMA mode** of the I2C2 interface for implementing read and write operation.

**Requirements:**

* Use the **I2C2 interface** to read data from the digital sensor
* Use the **UART4** interface to send data to the serial monitor
* Use the **interrupt mode** for reading/writing data
* Use the display format mentioned in the previous task

## D.1. Implementation

*Describe your solution. Add all relevant code snippets into a listing or as screenshots and describe their purpose.**Remember to use generic methods with well-defined parameters!!*

## D.2. Discussion

*Describe your experiences. Does your solution have any limitations? How would an ideal solution behave in your opinion? How is this task different from Task C?*

# Index

AHB *Avanced High-Performance Bus*

APB *Advanced Peripheral Bus*

CPU *Central Processing Unit*

DMA *Direct Memory Access*

RAM *Random Access Memory*

SRAM *Static-RAM*

TCM *Tightly Coupled Memory*

## Figures

[Figure 1: Interconnection 3](#_Toc169657891)

[Figure 2: Simplified DMA Interconnection Diagram 3](#_Toc169657892)

## Code Segments

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

## Tables

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

1. Tightly Coupled Memories (TCM) are dedicated memories directly connected to the processor, but not through a bus, so avoiding arbitration and latencies for frequently executed code. [↑](#footnote-ref-2)