**Networked Embedded Systems**

**Practicum 1: Accessing GPIOs**

**Group number: 8**

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

**Friday, April 19, 2024**

Map of Content

[Theory Questions 4](#_Toc164464943)

[Task A: Blink an LED 5](#_Toc164464944)

[a) Calculations 5](#_Toc164464945)

[b) Implementation 5](#_Toc164464946)

[c) Results 6](#_Toc164464947)

[d) Discussion 6](#_Toc164464948)

[Design Decisions 6](#_Toc164464949)

[Code Reusability 6](#_Toc164464950)

[Task B: Vary the Cycle Time 7](#_Toc164464951)

[a) Calculations 7](#_Toc164464952)

[b) Implementation 7](#_Toc164464953)

[c) Discussion 7](#_Toc164464954)

[Task C: Vary the Duty Cycle 8](#_Toc164464955)

[a) Calculations 8](#_Toc164464956)

[b) Implementation 8](#_Toc164464957)

[c) Discussion 8](#_Toc164464958)

[Task D: Vary the Blinking Frequency with Push Buttons 9](#_Toc164464959)

[a) Implementation 9](#_Toc164464960)

[b) Discussion 12](#_Toc164464961)

[Approach And Initial Thoughts 12](#_Toc164464962)

[Limits 12](#_Toc164464963)

[Possible Solutions 12](#_Toc164464964)

[Issues 12](#_Toc164464965)

[Task E: Display the Current Steps 13](#_Toc164464966)

[a) Calculations 13](#_Toc164464967)

[c) Implementation 14](#_Toc164464968)

[b) Results 16](#_Toc164464969)

[d) Discussion 16](#_Toc164464970)

[Bit Math 16](#_Toc164464971)

[Index 17](#_Toc164464972)

[Figures 17](#_Toc164464973)

[Code Segments 17](#_Toc164464974)

[Tables 17](#_Toc164464975)

# Theory Questions

1. **What does the abbreviation GPIO mean?**

|  |  |
| --- | --- |
| GPIO is short for **G**eneral **P**urpose **I**nput **O**utput and describes one of the most basic pin functionalities of a microcontroller besides other basic pin-functions such as:   * Power Pins (Beige) * GPIO Pins (Green or Gray) * Strapping Pins (Olive Green) | Ein Bild, das Rechteck, Text, Muster, Screenshot enthält.  Automatisch generierte Beschreibung  Figure 1: Pinout of the STM32H7 |

1. **What functionality do GPIOs have and what are they used for?**

**Analog IO**

Due to the digital nature of a microcontroller, reading and writing signals with voltage levels other than 3.3V or 0V requires a GPIO to be internally connected to a Digital-Analog-Converter (DAC) for outputs and an Analog-Digital-Converter (ADC) for inputs.

**Timers**

Timersare used for timing sensitive signals (e.g. PWM). A GPIO connected to a channel of a timer can be used to output this signal.

**Connectivity**

Certain GPIOs support various interfaces such as I2C, UART […], making it possible to connect peripherals like sensors (e.g. Gyroscope) or data storage (e.g. SD-Card) to these special GPIO-Pins.

**Interrupts**

GPIOs are capable of detecting edges and sending a flag to the interrupt-controller. The interrupt request can then be used to immediately execute a specified code.

1. **What is a duty cycle? How do you calculate it?**

|  |  |
| --- | --- |
| The duty cycle is the percentage of a period, in which the signal level is high. To calculate the duty cycle, the formular can easily be derived from Figure 2: | Figure 2: Duty Cycle of a Signal |

# Task A: Blink an LED

## Calculations

|  |  |
| --- | --- |
| **Cycle Time** | 3000ms |
| **Duty Cycle** | 66.6% |
| **LED ON Time** | 2000ms |
| **LED OFF Time** | 1000ms |

Table 1: Task A calculations

## Implementation

A function has been implemented to flash the decimal point of the segment display with a given period and a duty cycle. *HAL\_GPIO\_WritePin()* sets the GPIO output of a certain pin, in this case *SEGDP\_Pin* which is defined in *main.h* generated with CubeMX. *HAL\_Delay()* waits for a given time in milliseconds.

**

Codesegment 1: USR\_SEG\_FlashDP()

The Function is called indefinitely in the main while loop.



Codesegment 2: Relevant Code in main() for Task A

## Results

As a result, the decimal point of the seven segment display flashes.

|  |  |
| --- | --- |
| Ein Bild, das Elektronik, Elektronisches Bauteil, Schaltung, Elektrisches Bauelement enthält.  Automatisch generierte Beschreibung  Figure 3: Blinking Decimal Point (on) | Ein Bild, das Elektronik, Elektronisches Bauteil, Schaltung, Elektrisches Bauelement enthält.  Automatisch generierte Beschreibung  Figure 4: Blinking Decimal Point (off) |

## Discussion

### Design Decisions

Following the “*Pascal\_Snake”*-Casing of the HAL-Library code for a coherent code style and following the naming convention of *Library\_Module\_Function().*

|  |  |  |
| --- | --- | --- |
| Example | CubeMX Code | HAL\_GPIO\_Write(…) |
| Custom Code | *USR\_SEG\_FlashDP(…)* |

Furthermore, easy to read abbreviations are used for compact function and variable names.

### Code Reusability

The code has been consolidated into a function so it can be used in later tasks. If LEDs on pins other than the *SEGDP\_Pin* are to be “Flashed”, it would be clever to include GPIO Port and Pin into the function arguments.

The function could look like:



Codesegment 3: Alternative Flash Function

# Task B: Vary the Cycle Time

## Calculations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **LOW** | **MID** | **HIGH** | **Generic** |
| **Cycle Time** | 1000 | 500 | 200 | period\_ms [ms] |
| **Duty Cycle** | 66.6 | 66.6 | 66.6 | dc [%] |
| **LED ON Time** | 666 | 333 | 133 | [ms] |
| **LED OFF Time** | 333 | 167 | 66 | [ms] |

Table 2: Task B calculations

## Implementation

Defines for the blinking intervals:



Codesegment 4: Defines for Task B

The outer for-loop is used to iterate through the levels and the inner for-loop repeats the function call of *USR\_SEG\_FlashDP()* 15 times. This time an array is used to store the cycle time of the individual levels.



Codesegment 5: Relevant code in main() for Task B

## Discussion

Defines are used to reduce the amount of magic numbers in the code which makes it easier to read and understand. The *USR\_SEG\_FlashDP()* function can be adapted from Task A: Blink an LED.

# Task C: Vary the Duty Cycle

## Calculations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Step 1** | **Step 2** | **Step 3** | **Generic** |
| **Cycle Time** | 2000 | 2000 | 2000 | period\_ms [ms] |
| **Duty Cycle** | 70 | 50 | 30 | dc [%] |
| **LED ON Time** | 1400 | 1000 | 600 | [ms] |
| **LED OFF Time** | 600 | 1000 | 1400 | [ms] |

Table 3: Task C Calculations

## Implementation

The outer for-loop is used to iterate through the levels and the inner for-loop repeats the function call of *USR\_SEG\_FlashDC()* 15 times. This time an array is used to store the duty cycle of the individual levels.



Codesegment 6: Relevant code in main() for Task C

## Discussion

All defines, as well as the function *USR\_SEG\_FlashDP()* can be reused from Task B: Vary the Cycle Time.

# Task D: Vary the Blinking Frequency with Push Buttons

## Implementation

The values for defined constants are taken directly from the task sheet.

One goal is to keep the cycle time between 100 and 1000 milliseconds. The *LIMIT(x, min, max)* macro returns the input value *x* capped to the *min* or *max* value. The reason for a macro implementation instead of a function implementation is, that the macro takes all datatypes as opposed to a function.

Keep in mind that this macro can easily be exploited when directly calling a function within this macro (e.g. *LIMITS(Get\_Next()*[[1]](#footnote-1), …)) because *x* is directly replaced with the function and would be called three times. The macro could be reinforced with an extra declaration for *x* within the macro itself but won’t be implemented due to simplicity.



Codesegment 7: Task D defines and macros.

max

min

LIMIT(x, min, max)

Ein Bild, das Reihe, Diagramm, Screenshot, Rechteck enthält.

Automatisch generierte Beschreibung

x

Figure 5: LIMIT() Macro-Output

|  |  |
| --- | --- |
| The data we need to preserve to perform queries of a button is stored in a button-datatype with the fields shown in Codesegment 8.  To instance a button, a *Button\_t* declaration can be passed into the initialization function (Codesegment 9) to set the individual fields. | Codesegment 8: Button\_t type |



Codesegment 9: USR\_BTN\_Init()

A feature of the board is that the buttons are *active low*. This means that if a button is read while it is **not pressed**, it would return **high[[2]](#footnote-2)**. Therefor we need to **invert** the condition of a logical falling edge to get a physical falling edge (releasing the button).

|  |  |  |  |
| --- | --- | --- | --- |
| btn->prev | curr | Edge (logical) | btn->has\_changed |
| 0 | 0 | No change | 0 |
| 0 | 1 | Rising edge | 1 |
| 1 | 0 | Falling edge | 0 |
| 1 | 1 | No change | 0 |

Table 4: Truth Table for a physical falling edge



Codesegment 10: USR\_BTN\_Check()

In the main program, two instances of a button are created and initiated. The Flashing function is called in the while loop as before. Additionally, each of the buttons are read and checked for any changes. Because the *has\_changed* field of the Button can only be one ore zero, we can determine if the current period in- or decreases by applying a sign to the *step\_up* (+) and *step\_dn* (-) buttons *has\_changed* property.

By adding them together, we know if only one of the buttons has been pressed, otherwise *sgn* is 0. In theory the *if(sgn == 0)…* is not needed because the current cycle time would be incremented by 0, but it allows for an early return to save operations (more important in Task E: Display the Current Steps).



Codesegment 11: Relevant code in main() for Task D

## Discussion

### Approach And Initial Thoughts

“How to check if a button state is changed?”

To know if a button state is changed, the newly read button state has to be compared to the previous one. If they differ the button state changed.

“Considerations for rising and falling edge detections.”

Ignoring *how* the button changed would result in the cycle time being update twice. Once when pressed and once when released. The task sheet requires the cycle time to update when physically *releasing* the button.

“How should the button ‘module’ to look like?”

Implementing the necessities above needs a custom data structure for a button to save the previous button state and if the button has changed among other things to reduce the number of parameters in function calls and provide readable code.

### Limits

The issue with this implementation is, that a button is only checked after a blinking interval of the LED and does therefore not respond during a blinking cycle. With fast flashing the delay is barely noticeable but on slow delays you must hold down the button a considerable amount of time until the frequency updates.

To create a delay, *HAL\_Delay()* does nothing more than wait a certain time before continuing. This causes the LED to flash, but also prevents the *USR\_BTN\_Check()* function to be called during this wait.

### Possible Solutions

The ideal behavior would be to update the blinking frequency with a short push on the button. This can be achieved with either external interrupts on the buttons or handling the LED blinking by a timer.

### Issues

A potential issue would have been not noticing that the .ioc File does not configure the buttons as inputs.

# Task E: Display the Current Steps

## Calculations

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **SEGA** | **SEGB** | **SEGC** | **SEGD** | **SEGE** | **SEGF** | **SEGG** |
| **0** | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| **1** | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **2** | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| **3** | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| **4** | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| **5** | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| **6** | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| **7** | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **8** | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **9** | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

Table 5: Pin mask of the segment numbers

Calling *HAL\_GPIO\_WritePin()* for each LED on the display would be very cumbersome. Therefor a function is implemented, which writes directly to the port of the segment display. This simplification only works if all LEDs of the segment display are connected to the same GPIO port. In this case port J.

Using bit math, a map in form of an array is created which has also the convenience of returning the number mask of the index written in the brackets. In cases where there are less zeros than ones, the low pins are subtracted from the segment pin mask to shorten the expression.



Codesegment 12: Pin Mask Map

## Implementation

The defined constants are updated to match the task sheet requirements. Additional constants for the Segment display are defined as well.



Codesegment 13: defines

|  |  |
| --- | --- |
| To pass information into the segment display related functions, another datatype is created.  All parameters are assigned initial values in the init function. To save one parameter in the function call, the mask of the number 8 can be used to set the *pin\_mask* variable, as it lights up all the LEDs. | Codesegment 14: Segment type |

**

Codesegment 15: USR\_SEG\_Init()

The write function for the seven segment limits the input number to 0 through 9 and modifies the register according to the predefined number mask array. This is also an example of reusability, as the LIMIT macro is used here as well.

**

Codesegment 16: USR\_SEG\_Write()

The only thing added to the main function is a *Segment\_t* instance as well as the write function after a button press is detected. In the call of the write function, the number is incremented by the in Task D: Vary the Blinking Frequency with Push Buttons documented *sgn* variable.

**

Codesegment 17: Relevant code in main for Task E

## Results

Following figures show various states of the program.

|  |  |  |
| --- | --- | --- |
| Ein Bild, das Elektronik, Text, Elektronisches Bauteil, Elektrisches Bauelement enthält.  Automatisch generierte Beschreibung |  |  |

Figure 6: Results Task E

## Discussion

As mentioned in Limits from Task E, updating the cycle requires one to hold down a button for a longer period of time.

### Bit Math

Useful bit operations:



Codesegment 18: Bitmath macros

# Index

## Figures

[Figure 1: Pinout of the STM32H7 4](#_Toc164464976)

[Figure 2: Duty Cycle of a Signal 4](#_Toc164464977)

[Figure 3: Blinking Decimal Point (on) 6](#_Toc164464978)

[Figure 4: Blinking Decimal Point (off) 6](#_Toc164464979)

[Figure 5: LIMIT() Macro-Output 9](#_Toc164464980)

[Figure 6: Results Task E 16](#_Toc164464981)

## Code Segments

[Codesegment 1: USR\_SEG\_FlashDP() 5](#_Toc164464982)

[Codesegment 2: Relevant Code in main() for Task A 5](#_Toc164464983)

[Codesegment 3: Alternatie Flash Function 6](#_Toc164464984)

[Codesegment 4: Defines for Task B 7](#_Toc164464985)

[Codesegment 5: Relevant code in main() for Task B 7](#_Toc164464986)

[Codesegment 6: Relevant code in main() for Task C 8](#_Toc164464987)

[Codesegment 7: Task D defines and macros. 9](#_Toc164464988)

[Codesegment 8: Button\_t type 10](#_Toc164464989)

[Codesegment 9: USR\_BTN\_Init() 10](#_Toc164464990)

[Codesegment 10: USR\_BTN\_Check() 10](#_Toc164464991)

[Codesegment 11: Relevant code in main() for Task D 11](#_Toc164464992)

[Codesegment 12: Pin Mask Map 13](#_Toc164464993)

[Codesegment 13: defines 14](#_Toc164464994)

[Codesegment 14: Segment type 14](#_Toc164464995)

[Codesegment 15: USR\_SEG\_Init() 14](#_Toc164464996)

[Codesegment 16: USR\_SEG\_Write() 15](#_Toc164464997)

[Codesegment 17: Relevant code in main for Task E 15](#_Toc164464998)

[Codesegment 18: Bitmath macros 16](#_Toc164464999)

## Tables

[Table 1: Task A calculations 5](#_Toc164465000)

[Table 2: Task B calculations 7](#_Toc164465001)

[Table 3: Task C Calculations 8](#_Toc164465002)

[Table 4: Truth Table for a physical falling edge 10](#_Toc164465003)

[Table 5: Pin mask of the segment numbers 13](#_Toc164465004)

1. This is an example function, sensitive to multiple consecutive calls. [↑](#footnote-ref-1)
2. This is preferred due to current consumption of the GPIO driver. [↑](#footnote-ref-2)