Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Laborbericht 1

I/O Ports

Ausarbeitung Gruppe 20

Laborbericht 1 eingereicht von
Kühne, Sebastian
Matrikelnummer 2604332
Laborgruppe 20

im Rahmen der Vorlesung *INF3 S22*
im Studiengang Media Systems
am Department Medientechnik der Fakultät DMI
an der Hochschule für Angewandte Wissenschaften Hamburg

Lehrende*r: Prof. Dr. Tessa Taefi

Eingereicht am: 18.04.2022

Zusammenfassung

Im Rahmen der ersten Labor-Übung des Kurses „Informatik 3 und Elektronik“ soll ein Atmel „Atmega 328P“ Board zusammen mit einem 10-LED Shield über selbstentwickelten C Code in Betrieb genommen werden.
Diese wird im Vorhinein mithilfe von Schaltplänen und UML Aktivitätsdiagrammen geplant und in der Entwicklungsumgebung „Microchip Studio“ getestet, um im Anschluss entsprechende Erkenntnisse wissenschaftlich zu dokumentieren.

Inhaltsverzeichnis

ABKÜRZUNGSVERZEICHNIS

MCS	Microchip Studio

ABBILDUNGSVERZEICHNIS

FORMEL VERZEICHNIS

1 LABOR 1 - I/O PORTS

Dieser Laborbericht entsteht als Teil einer Prüfungsleistung im Fach “Informatik und Elektronik”, in dessen Rahmen die Aufgaben des Labor 1 bearbeitet, dokumentiert und geprüft werden sollen. Die Abnahme erfolgt über eine praktische Vorstellung der Ergebnisse während einer Laboreinheit.

Geprüft werden das Verständnis um die Funktionsweise des Atmel 328P und dem Programm Microchip Studio (fortlaufend „MCS“).

Dabei sollen der Atmel 328 und das selbstentwickelte LED Shield in Betrieb genommen werden. Selbst geschriebene, den Anforderungen der Aufgaben entsprechende Firmware soll die I/O Ports des Controllers ansteuern.

Ein weiteres Ziel ist das Verständnis und die Nutzung von Interrupts.

1.1 ZIELE DER LABORAUFGABEN

Durch das Ausführen der selbstgeschriebenen Firmware auf dem ATmega 328P und dem Shield sollen die, in der Vorlesung erlernten Kenntnisse demonstriert werden.

Dabei sollen Programme geschrieben werden, die verschiedene Funktionen des Boards nutzen und auftretende Probleme analysiert und behoben werden.

1.2 ANFORDERUNGSANALYSE

Das Labor 1 stellt folgende Anforderungen:

Aufgabe 1: Xplained Mini Board

“LEDOnButtonPress” soll auf dem Controller ausgeführt werden. Dieses Programm soll eine LED aufleuchten lassen, wenn ein Push-Down-Butten betätigt wird.

Anschließend soll “CountButtonAndBlink” entwickelt und ausgeführt werden. Diese ist eine Erweiterung von “LEDOnButtonPress” und soll zählen, wie oft der Button bereits betätigt wurde und die angeschlossene LED dem entsprechend oft blinken lassen.

Aufgabe 2: Shield in Betrieb nehmen

Nachdem berechnet wurde, ob das Board genug Strom für das Shield liefern kann, soll "LEDOnButtonPress" auf das Shield portiert und durch Button 1 gesteuert werden. Es soll ebenfalls geprüft werden, was passiert, wenn der Pull-Up-Widerstand des Buttons entfernt wird.

Durch eine Erweiterung des Programms soll Button 2 die Funktion bekommen die Anzahl der Buttonpresses ("CountButtonAndBlink") zurückzusetzen.

Der letzte Teil dieser Aufgabe fordert, dass die Anzahl der Bottonpresses durch 4 LEDs auf dem Shield als Dualzahlen dargestellt werden. Diese soll in 4 Bit dargestellt werden können und beim Overflow auf 0 zurückgesetzt werden. Button 2 behält die Funktion, die Anzahl der Buttonpresses zurückzusetzen.

Aufgabe 3: Interrupts

Im Abschließenden Teil der Laborübung sollen jeweils alle Grünen bzw. Roten LEDs des Shields mithilfe der beiden Buttons gesteuert werden.
Dies soll zuerst, wie in den vorherigen Aufgaben, durch Polling erreicht werden, was anschließend durch Interrupts ersetzt werden soll.

2 LABORVORBEREITUNG UND METHODIK

Sämtliche Aufgaben wurden unter MS Windows 10 und teilweise über ein durch USB verbundenes Atmel 328P Board ausgeführt, dazu wurden folgende Software und Datensätzen genutzt.

Blockschaltbilder und Ablaufdiagramme wurden in MS Visio erstellt.

Software wurde anhand vorher erstellter Ablaufdiagramme in MCS entwickelt, welches ebenfalls als Hauptwerkzug zum Testen und Debugging verwendet wurde.

Weitere Tests und Erstellung von Schaltplänen erfolge in SimulIDE

MCS wurde aufgrund der tiefen Einbindung aller Atmel Produkte gewählt, da so einfaches Simulieren der Software sowie Hardware-Debugging direkt aus der IDE möglich war.

SimulIDE bietet eine Alternative Firmware zu testen sofern kein Chip zur Hand ist und ermöglicht aufgrund der Vorkonfiguration verschiedener Microcontroller einfaches Prototyping von Schaltungen und eine schnelle Erstellung von Schaltplänen.

Die Ausführung und Vorbereitung des Labors wurden durch die Verfügbarkeit von Microcontrollern außerhalb der Laborzeiten limitiert, was hauptsächlich durch Simulation in SimulIDE ergänzt werden konnte.

3 DOKUMENTATION DER LABORARBEIT

Im Folgenden wird die Planung und Durchführung der „Laboreinheit 1: I/O Ports“ dokumentiert. Dies wird analog zu den von Prof. Dr. Taefi zur Verfügung gestellten Folien in drei Unterpunkte gegliedert.
Des Weiteren folgen jeweils Planung und Vorbereitung des Labors separat von Implementation und Beobachtung.

3.1 PLANUNG DER LABORAUFGABEN

Sämtliche Laboraufgaben wurden im Vorhinein durch Material aus der Vorlesung „Informatik 3 und Elektronik“ und Methoden aus Kapitel 2 vorbereitet und konnten erst zum Labortermin getestet und abgenommen werden.

3.1.1 Xplained Mini Board

Laboraufgabe 1A erfordert das Laden eines zuvor geschriebenen Programms „LEDOnButtonPress“ auf den Controller, welches durch Betätigen des Push-Down Buttons (siehe Schaltplan Abbildung 1) eine angeschlossene LED kontrolliert.

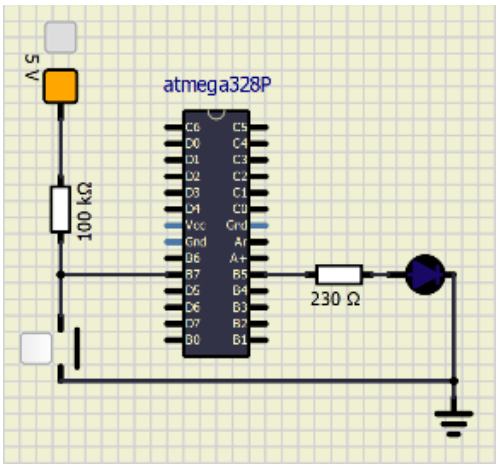

Abbildung 1: Schatplan Atmega 328P mit button und LED

Das Setup von Button, Controller und LED kann hierbei durch das Blockschaltbild in Abbildung 2 beschrieben werden, wobei der Input des Tasters durch den Controller verarbeitet und anschließend als Output an eine LED weitergegeben wird.

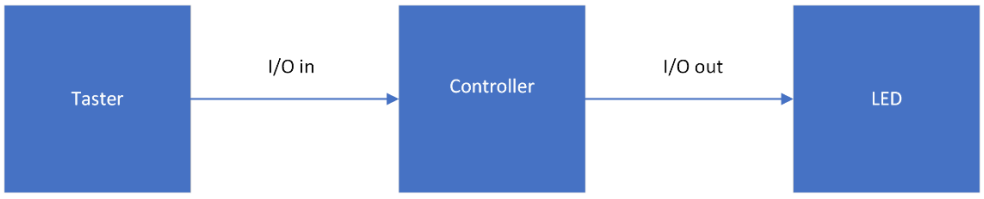

Abbildung 2: Blockschaltbild LEDOnButtonPress

Die Handhabung des Inputs sowie die funktionalität der Software wird durch den Ablaufplan in Abbildung 3 beschrieben, welche ebenfalls aufzeigt das es sich hierbei um einen unendlich Kreislauf handelt, in welchem das Programm auf Nutzerinput prüft und entsprechend reagiert.

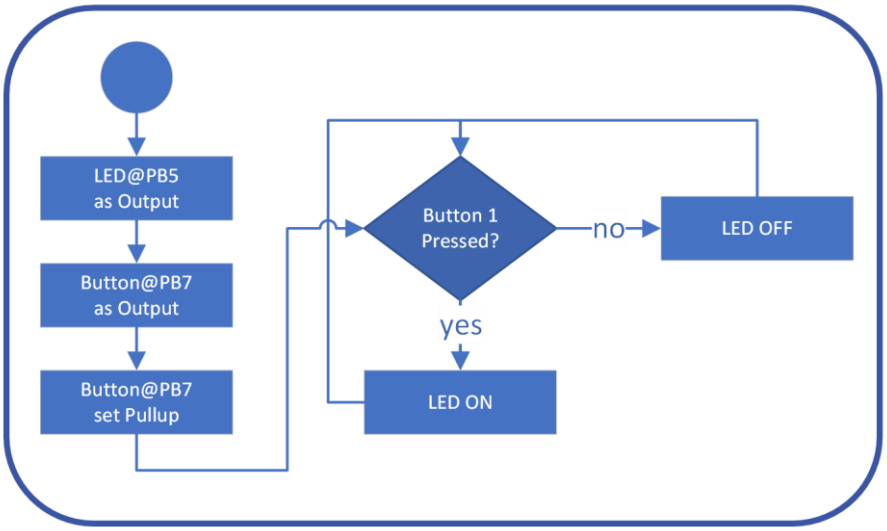

Abbildung 33: UML Aktivitätsdiagram LEDOnButtonPress

Nach dem erfolgreichen Flashen des Controllers mithilfe von MCS, soll anschließend in *Aufgabe 1B* ein komplexeres Programm entwickelt werden. „CountButtonAndBlinK“ verwendet das selbe in Abbildung 2 beschriebene Hardware Setup wie „OnButtonPress“ und wird über den Ablaufplan in Abbildung 4 veranschaulicht.

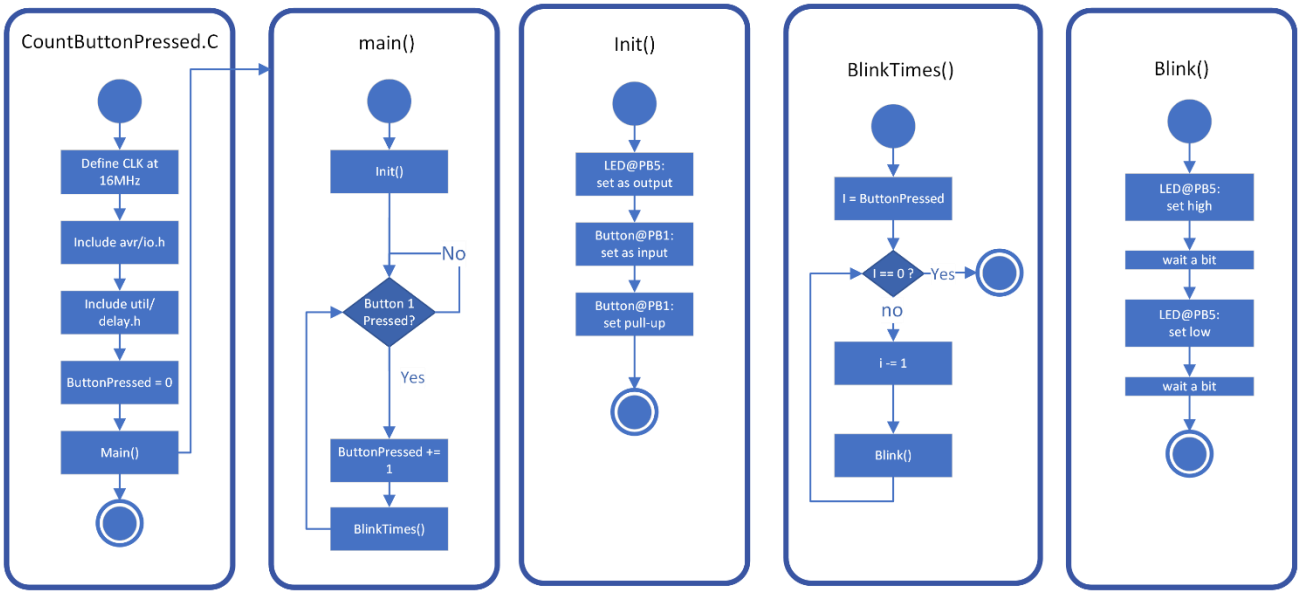

Abbildung 4: UML Aktivitätsdiagram CountAndBlink

Das Programm speichert zuerst einen Wert „ButtonPressed“, welcher fortlaufend mit jedem Drücken des Tasters steigt.
Nach jedem Drück des Tasters blinkt die zuvor verwendete LED nun so häufig wie der Taster gedrückt wurde, wobei auch hier wieder ein endlos auf Nutzerinput wartender Kreislauf den Hauptteil des Programms darstellt.

3.1.2 Shield und I/O Ports

Wie in *Laboraufgabe 2A* beschrieben, soll vor der Inbetriebnahme des Shields geprüft werden, ob der Controller genug Strom für das 10 Segment LED-Display des Shields liefert.
Mithilfe des in Abbildung 5 gezeigten Schaltplanes und dem in Abbildung 6 gezeigten Datenblatt des LED Displays, lassen sich folgende Werte ermitteln:

V_F	Forward Voltage	Bright Red High Efficiency Red Green Yellow Super Bright Red	2.0 2.0 2.2 2.1 1.85	2.5 2.5 2.5 2.5 2.5	V	IF=20mA

Abbildung 5: Auszug Knightbrigth DC-10E Datasheet

V = 5 Volt | R = 220 Ohm | U_R Green = 5 – 2.2 = 2.8V | U_R Bright Red = 5 – 2 = 3V

$$Green = \frac{2.8V}{220Ohm} = 0.01\overline{27}A$$

$$Bright\ Red\ = \frac{3V}{220Ohm} = 0.01\overline{36}A$$

1 Ohm's Law

Mit sieben LEDs vom Typ „Green“ á 12mA, und drei vom Typ „Bright Red“ á 13mA lässt sich der Gesamtbedarf des Shields auf ca. 123mA berechnen. Somit überschreitet der Gesamtbedarf des Shields nicht das im ATmega328P angegebene Maximum von 150mA für Ports, und ist mit 20mA pro Output Pin pro LED ebenfalls gedeckt.

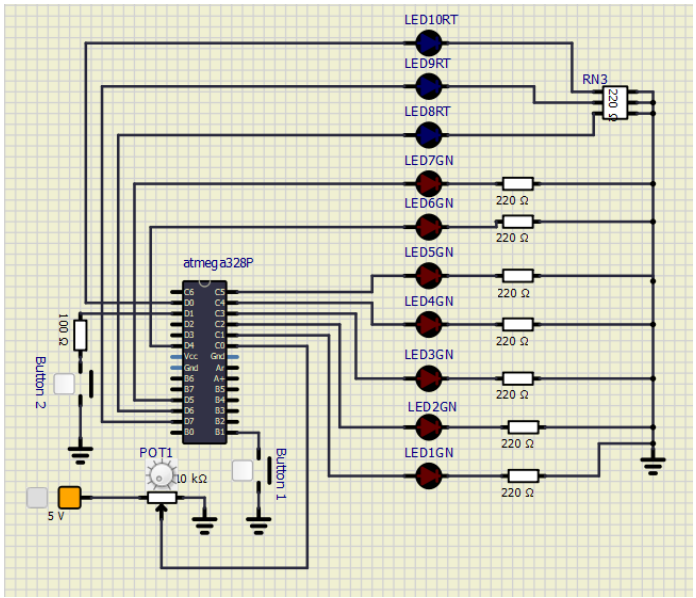

Abbildung 6: Schaltplan Atmega 328P mit selbstentwickeltem LED Shield

Laboraufgabe 2B beschreibt, wie nun nach dem Verbinden des Shields das in Abbildung 3 beschriebene Programm „LEDOnButtonPress“ entsprechend auf das Shield angepasst werden soll.

Hierzu ändert sich in der Planung lediglich das Blockschaltbild, da nun das 10-Segment-Led anstelle der internen LED als Output dient.

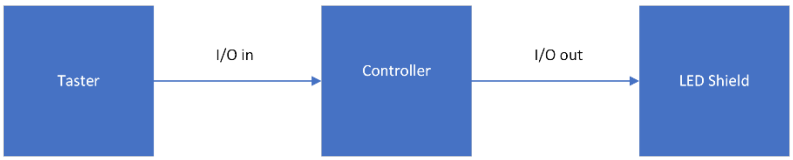

Abbildung 7: Blockschaltbild Atmega 328P mit Shield

Die in Abbildung 7 zu sehende Veränderung zu Abbildung 2 wird so auch in der Software durch eine Änderung der In und Output Ports widergespiegelt.

Weiter wird durch *Laboraufgabe 2D* ein zweiter Button zum zurücksetzen des in Abbildung 4 beschriebenen Programmes „CountButtonAndBlink“ eingeführt, was sich im nun Angepassten Blockschaltbild in Abbildung 7 zeigt, welches nun den Input der zwei Buttons verarbeitet und an das LED Shield weitergibt.

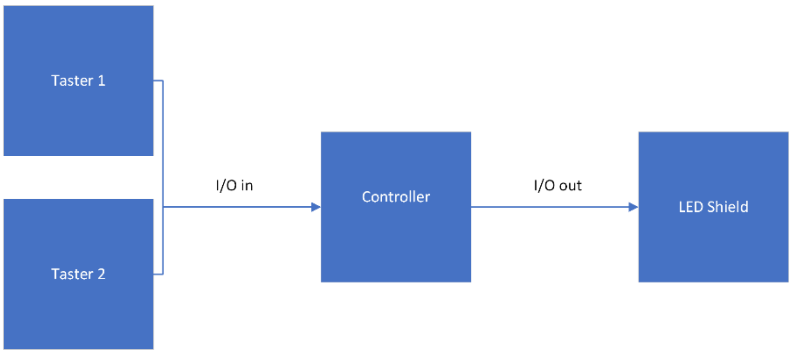

Abbildung 8: Blockschaltbild CountButtonAndBlink mit Shield

Die Einführung des zweiten Buttons und dem damit verbundenen Zurücksetzen des Counters, lässt sich durch eine zweite „wenn-dann“ Verzweigung im Ablaufplan in Abbildung 10 beschreiben.

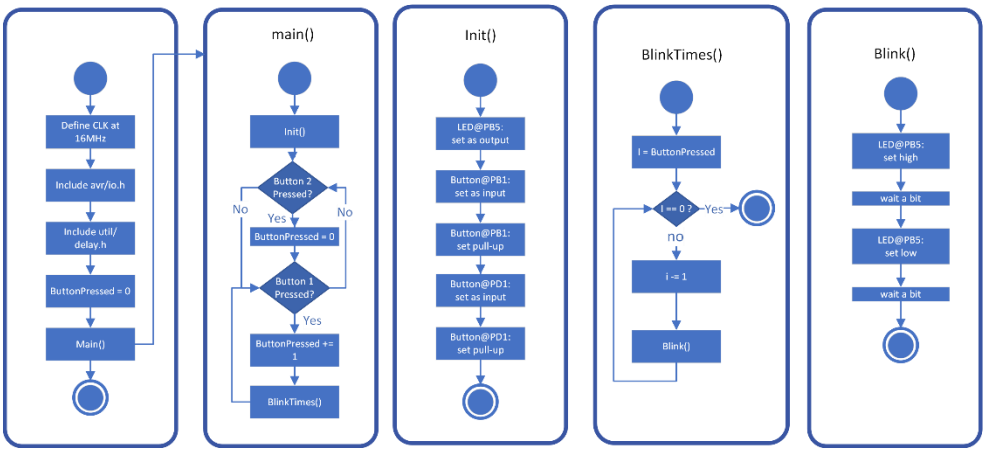

Abbildung 9: UML Aktivitätsdiagram CountButtonAndBlink

Somit wird die Nutzereingabe permanent auf das drücken des zweiten Buttons abgefragt, und sollte dieser gedrückt werden, der Counter auf 0 gesetzt.

In *Laboraufgabe 2E* soll das Programm schließlich so angepasst werden, dass das LED Shield nun den Counter als Dualzahl wiederspiegelt. Dies wird erreicht indem der Counter einfach das Output-Register des Controllers gesetzt, und an die entsprechende Stelle verschoben wird.

Da der Counter allerdings auf Button 1 steigt, ist wie im in Abbildung 11 gezeigten Ablaufplan ebenfalls eine Software Lösung in Form einer „während“ Schleife zum entprellen des Buttons angelegt. Dies ist für Button 2 nicht notwendig, da das wiederholte Zurücksetzen auf 0 keine weiteren Auswirkungen hat.

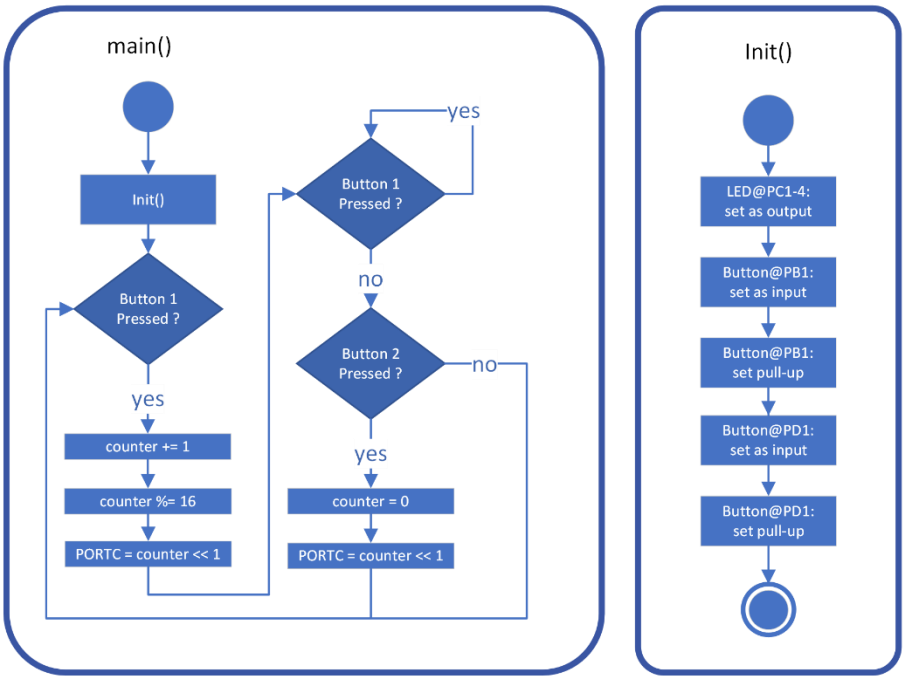

Abbildung 10: UML Aktivitätsdiagram CountButtonShowBinary

3.1.3 Interrupts

In *Aufgabe 3A* sollen jeweils alle Grünen, bzw. alle Roten LEDs des Shields entsprechend über Button 1 und Button 2 gesteuert werden. Abbildung 11 Beschreibt hier den Ablauf des Programms, welches Nutzernput pollt, also endlos Input abfragt um entsprechende Toggle zu aktivieren.

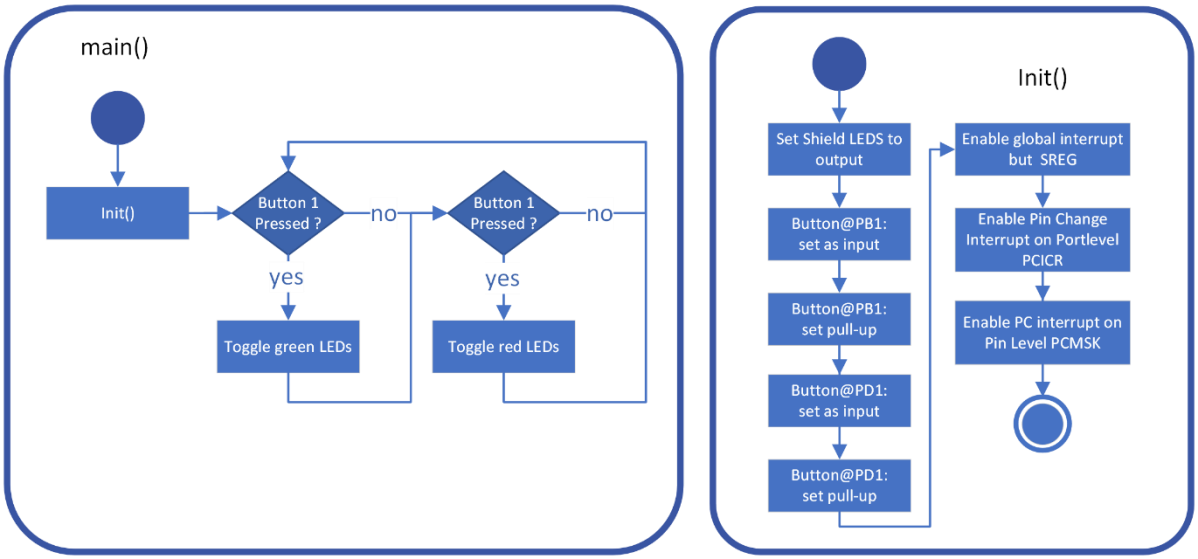

Abbildung 11: UML Aktivitätsdiagramm ToggleWithPolling

Aufgabe 3B stellt eine Alternative zum Polling vor und ersetzt das zuvor erwähnte Polling durch Interrupts, welche zuvor definiert und aktiviert werden müssen, aber so die ständige abfrage des Nutzerinputs ersetzen (siehe Abbildung 12).

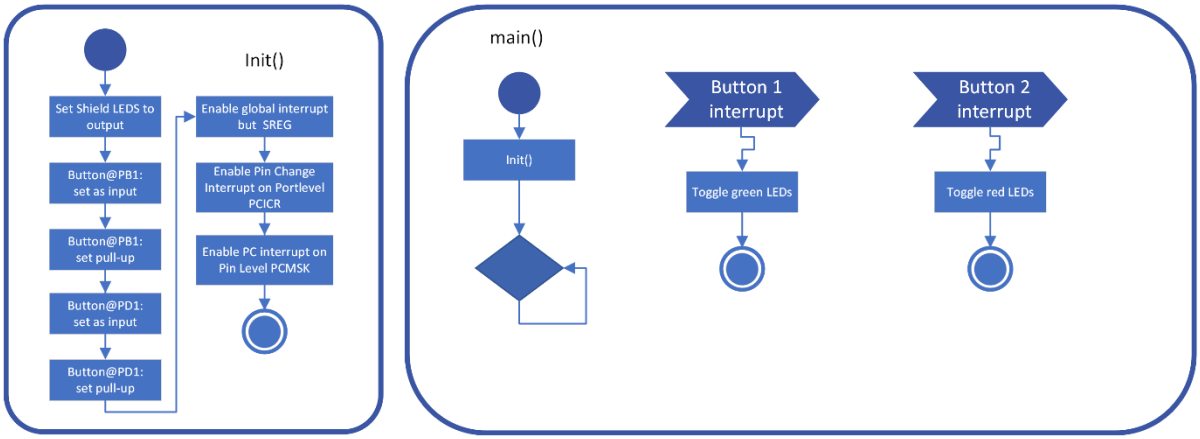

Abbildung 12: UML Aktivitätsdiagram ToggleLEDsWithInterrupts

3.2 Implementation und Testung

Implementation des zuvor geplanten Codes, sowie Tests aus Aufgaben wurden zum Labortermin mithilfe der Planung aus Kapitel 3.1 ausgeführt und anschließend dokumentiert.

3.2.1 Laden von selbstentwickelter Software auf das Xplained Mini Board

Zuvor erstellter Code konnte mithilfe von MCS problemlos auf das Board geflasht werden.
Abbildung 13 zeigt die gesamte main.c Datei für „LEDOnButtonPress“, in welcher die Board internen Buttons und LEDs über die entsprechenden Ports gesteuert werden.

```c
#include <avr/io.h>

#define LED_ON PORTB |= (1<<5)
#define LED_OFF PORTB &= ~(1<<5)
#define LED_TOGGLE PINB |= (1<<5)
#define BUTTON_PRESS (!(PINB & (1<<7)))

void init(void){
    // LED
    DDRB |= (1<<5); // Configure PB5 as Output
    LED_ON;

    // Button
    DDRB &= ~(1<<7);//Configure PB1 as Input
    PORTB |= 1<<7;  //Enable Internal Pull-Up at PB1
}

int main(void){
    init();
    while (1){
        if(BUTTON_PRESS)
            LED_ON;
        else
            LED_OFF;
    }
}
```

Abbildung 13: LEDOnButtonPress - main.c

Zu Aufgabe 1B Zeigt Abbildung 13 einen Ausschnitt der main.c datei für „CountButtonAndBlin“, welche die until\delay.h header datei nutzt um Pausen zwischen LED-Zustandswechseln zu setzen.

```c
void blink(unsigned n){
    for(unsigned i = 0; i <= n; i++){
        LED_ON;
        _delay_ms(1000);
        LED_OFF;
        _delay_ms(1000);
    }
}

int main(void){
    init();

    while (1){
        if(BUTTON_PRESS){
            blink(buttonPresses ++);
        }
    }
}
```

Abbildung 14: Ausschnitt CountButtonAndBlink - main.c

3.2.2 Inbetriebnahme eines selbstentwickelten LED Shields

Zu Aufgabe 2C konnte im Labor beobachtet werden, das durch das Deaktivieren des Pull-Ups der Button keine Funktion mehr aufzuweisen scheint, und das Board permanent aus ist wie in Abbildung 15 zu sehen ist.

Abbildung 15: Board mit aufgestecktem Shield

Zu Laboraufgabe 2E ist in der Implementation von main.c zu beachten, das statt einzelne Bits zu setzen hier einfach das gesamte Register PORTC gleich dem Counter, bzw. Null gesetzt wird (siehe Abbildung 16). Während des Labors konnte beobachtet werden das die Software ohne entprellen nicht nutzbar ist.

```
int main(void){
    init();

    while (1){
        if(BUTTON_ONE_PRESS){
            counter ++;
            counter %= 16;
            PORTC = (counter << 1);
            while(BUTTON_ONE_PRESS){_delay_ms(100);}
        }
        if (BUTTON_TWO_PRESS){
            counter = 0;
            PORTC = (counter << 1);
        }
    }
}
```

Abbildung 16: Ausschnitt main.c für CountButtonBinary

3.2.3 Nutzung von Interrupts auf dem Xplained Mini Board

Bei der Implementation von Aufgabe 3 konnte gezeigt werden wie das Setzen ganzer Register Code Übersichtlicher macht, wie in an den Makros in Abbildung 17 deutlich wird.

```
#define GREEN_ON {PORTC |= 0b00111110; PORTD |= 0b00110000;}
```

```
#define GREEN_OFF {PORTC &= ~0b00111110; PORTD &= ~0b00110000;}

#define RED_ON PORTD    |= 0b11000001
#define RED_OFF PORTD   &= ~0b11000001

#define BUTTON_ONE_PRESS (!(PIND & (1<<1)))
#define BUTTON_TWO_PRESS (!(PINB & (1<<1)))
```

Abbildung 17: Ausschnitt main.c für ToggleLED

4 FAZIT

Während der Verwendung verschiedener Software mit dem Board und Shield konnten verschiedene Registerverhalten beobachtet und verstanden werden. Dazu wurde durch die Ausarbeitung Inputsensitiver Software Prellen beobachtet und behoben, was ein besseres Verständnis des Zusammenhangs zwischen Hardware und Software ermöglichte.

Die Nutzung verschiedener Interrupts führte außerdem zu einem Besserem Verständnis der Funktionalität des ATmega 328P und ermöglichte einen Tangiblen Unterschied zwischen Polling und Interrupts festzustellen.

ANHANG

Quellcode 1A LEDOnButtonPress:

```
#include <avr/io.h>

#define LED_ON PORTB |= (1<<5)
#define LED_OFF PORTB &= ~(1<<5)
#define LED_TOGGLE PINB |= (1<<5)
#define BUTTON_PRESS (!(PINB & (1<<7)))

void init(void){
    // LED
    DDRB |= (1<<5); // Configure PB5 as Output
    LED_ON;

    // Button
    DDRB &= ~(1<<7);//Configure PB1 as Input
    PORTB |= 1<<7;  //Enable Internal Pull-Up at PB1
}

int main(void){
    init();
    while (1){
        if(BUTTON_PRESS)
            LED_ON;
        else
            LED_OFF;
    }
}
```

Quellcode 1B CountButtonAndBlink:

```
#define F_CPU 16E6          // or F_CPU 16000000

#include <avr/io.h>
#include <util/delay.h>

#define LED_ON PORTB |= (1<<5)
#define LED_OFF PORTB &= ~(1<<5)
#define LED_TOGGLE PINB |= (1<<5)

#define BUTTON_PRESS (!(PINB & (1<<7)))

unsigned buttonPresses = 0;

void init(void){
    // LED
    DDRB |= (1<<5);     // Configure PB5 as Output

    // Button
    DDRB &= ~(1<<7);    //Configure PB1 as Input
    PORTB |= 1<<7;      //Enable Internal Pull-Up at PB1
}

void blink(){
    for(unsigned i = 0; i <= buttonPresses; i++){
        LED_ON;
        _delay_ms(1000);
        LED_OFF;
        _delay_ms(1000);
    }
}

int main(void){
    init();

    while (1){
        if(BUTTON_PRESS){
            blink(buttonPresses ++);
        }
    }
}
```

Quellcode 2B CountOnShield:

```
#define F_CPU 16E6          // or F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>
#define LED_ON PORTC |= (1<<1)
#define LED_OFF PORTC &= ~(1<<1)
#define LED1GN_TOGGLE PINC |= (1<<1)
#define LED2GN_TOGGLE PINC |= (1<<2)
#define LED3GN_TOGGLE PINC |= (1<<3)
#define LED4GN_TOGGLE PINC |= (1<<4)

#define BUTTON_ONE_PRESS (!(PIND & (1<<1)))
#define BUTTON_TWO_PRESS (!(PINB & (1<<1)))

unsigned blinkTimes = 0;

void init(void){
    // LED
    DDRB |= (1<<1);     // Configure PB5 as Output

    // Buttons
    DDRB &= ~(1<<1);    //Configure PB1 as Input
    PORTB |= 1<<1;      //Enable Internal Pull-Up at PB1
    DDRD &= ~(1<<1);    //Configure PD1 as Input
    PORTD |= 1<<1;      //Enable Internal Pull-Up at PD1
}

void blink(){
    for(unsigned i = 0; i < blinkTimes; i++){
        LED_ON;
        _delay_ms(500);
        LED_OFF;
        _delay_ms(500);
    }
}

int main(void){
    init();

    while (1){
        if(BUTTON_ONE_PRESS){
            blink(blinkTimes ++);
        } else if (BUTTON_TWO_PRESS){
            blinkTimes = 0;
        }
    }
}
```

Quellcode 2E CountBinary:

```
#define F_CPU 16E6          // or F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>

#define BUTTON_ONE_PRESS (!(PIND & (1<<1)))
#define BUTTON_TWO_PRESS (!(PINB & (1<<1)))

unsigned counter;

void init(void){
    // counter
    counter = 0;

    // LED
    DDRC |= 0b00011110;     // Configure PC 1 - 4 as Output

    // Buttons
    DDRB &= ~(1<<1);    //Configure PB1 as Input
    PORTB |= 1<<1;      //Enable Internal Pull-Up at PB1

    DDRD &= ~(1<<1);    //Configure PD1 as Input
    PORTD |= 1<<1;      //Enable Internal Pull-Up at PD1
}


int main(void){
    init();

    while (1){
        if(BUTTON_ONE_PRESS){
            counter ++;
            counter %= 16;
            PORTC = (counter << 1);
            while(BUTTON_ONE_PRESS){_delay_ms(100);}
        }
        if (BUTTON_TWO_PRESS){
            counter = 0;
            PORTC = (counter << 1);
        }
    }
}
```

Quellcode 3A TogglePolling:

```
#include <avr/io.h>

#define GREEN_ON {PORTC |= 0b00111110; PORTD |= 0b00110000;}
#define GREEN_OFF {PORTC &= ~0b00111110; PORTD &= ~0b00110000;}

#define RED_ON PORTD    |= 0b11000001
#define RED_OFF PORTD   &= ~0b11000001

#define BUTTON_ONE_PRESS (!(PIND & (1<<1)))
#define BUTTON_TWO_PRESS (!(PINB & (1<<1)))

void init(void){
    // LEDS
    DDRC |= 0b00111110;
    DDRD |= 0b11110001;

    // Buttons
    DDRB &= ~(1<<1);    //Configure PB1 as Input
    PORTB |= 1<<1;      //Enable Internal Pull-Up at PB1

    DDRD &= ~(1<<1);    //Configure PD1 as Input
    PORTD |= 1<<1;      //Enable Internal Pull-Up at PD1
}

void check_buttons(void){
    if(BUTTON_ONE_PRESS)    {GREEN_ON;}
    else                    {GREEN_OFF;}

    if(BUTTON_TWO_PRESS)    {RED_ON;}
    else                    {RED_OFF;}
}

int main(void){
    init();
    while (1){
        check_buttons();
    }
}
```

QuellCode 3B ToggleInterrupt:

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define GREEN_ON {PORTC |= 0b00111110; PORTD |= 0b00110000;}
#define GREEN_TOGGLE {PORTC ^= 0b00111110; PORTD ^= 0b00110000;}
#define RED_ON {PORTD |= 0b11000001; PORTB |= (1<<0);}
#define RED_TOGGLE {PORTD ^= 0b11000001; PORTB ^= (1<<0);}

#define BUTTON_ONE_PRESS (!(PIND & (1<<1))) // -> PCINT17
#define BUTTON_TWO_PRESS (!(PINB & (1<<1))) // -> PCINT1

// Triggers on PD changes
ISR(PCINT2_vect) { GREEN_TOGGLE;}
// Triggers on PB changes
ISR(PCINT0_vect) { RED_TOGGLE;}

void init(void){

    // DDR
    DDRB |= 0b00000000;
    DDRC |= 0b00111110;
    DDRD |= 0b11110001;

    // Buttons
    PORTD |= 1<<1;
    PORTB |= 1<<1;

    GREEN_ON; // turn on LEDS
    RED_ON;

    // Pin change interrupt btn1
    sei();                  // enable global interrupt

    PCICR |= (1<<PCIE2);    // enable PD PC interrupt
    PCICR |= (1<<PCIE0);    // enable PB PC interrupt
    PCMSK0 |= (1<<1);   // mask out PB1 interrupt
    PCMSK2 |= (1<<1);   // mask out PD1 interrupt
}

int main(void){
    init();
    while (1){

    }
}
```

EIGENSTÄNDIGKEITSERKLÄRUNG

Hiermit versichern wir, dass wir das vorliegende Dokument selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die wir wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen haben, haben iwir deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

	Sebastian Kühne
19. April 2020	Nico Klärmann
Datum	Unterschrift