Hochschule
München
University of
Applied Sciences

Wintersemester 2021/22

Embedded Systems (Studiengänge MBB, FAB)

Schriftliche Fernprüfung mit Videoaufsicht

Prüfer: Küpper

Bearbeitungszeit: 90 Minuten

Hilfsmittel:

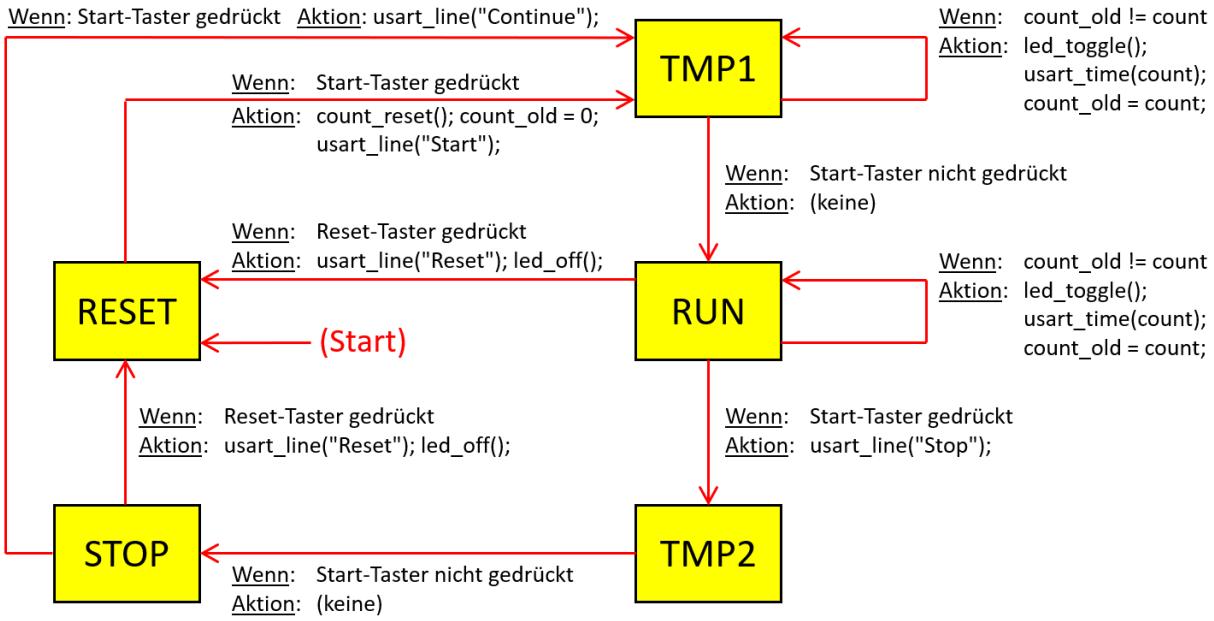
- **Taschenrechner sind zugelassen.**
- **Alle schriftlichen Unterlagen sind erlaubt.**
- **Der PC darf während der Prüfung nur zur Anzeige des Aufgabenblatts genutzt werden.**

Schreiben Sie Ihren Namen, Vornamen und auch die Studiengruppe auf alle Lösungsblätter. Es werden nur handschriftliche Lösungen auf leeren, weißen DIN-A4-Blättern akzeptiert.

****** Viel Erfolg! ******

Aufgabe 1 von 4 (Mikrocontroller-Programmierung, ca. 45 Punkte ≙ 45 Minuten)

Der folgende endliche Automat beschreibt das Verhalten einer Stoppuhr:



- Es wird ein ATmega328P-Mikrocontroller verwendet (Takt: 16 MHz, wie im Praktikum).
- Die Stoppuhr verfügt über einen Start-Taster an PB3, einen Reset-Taster an PD7 und eine Leuchtdiode (LED) an PC0. Wenn ein Taster gedrückt wird, dann verbindet er den jeweiligen Eingang des Mikrocontrollers mit Masse/GND.
- An beiden Tastern sind die internen Pullup-Widerstände zu aktivieren.
- Die Ausgabe der Stoppuhr (also zum Beispiel die aktuelle Zeit) erfolgt über die serielle Schnittstelle des Mikrocontrollers.

Aufgaben:

1.1. Wie lauten die fehlenden Befehle in der Funktion `void init_ports(void)` zur Initialisierung der Ein- und Ausgänge PB3, PD7, PC0 des Mikrocontrollers?

1.2. Wie lauten die fehlenden Befehle in den Funktionen `button_start()` und `button_reset()`? Diese Funktionen geben eine 1 zurück, falls der jeweilige Taster gerade eingeschaltet (gedrückt) ist, ansonsten ist die Rückgabe gleich 0. Die Funktionen sollen also nicht warten, bis der jeweilige Taster wieder losgelassen wird, sondern das Ergebnis sofort zurückgeben. Es soll auch nichts „entprellt“ werden ...

1.3. Ist die Leuchtdiode im Zustand STOP an oder aus? (Kurze Begründung!)

1.4. Wie oft wird `ISR(TIMER1_COMPA_vect)` pro Sekunde aufgerufen, wenn die Stoppuhr läuft?

1.5. Mit welcher Frequenz blinkt die Leuchtdiode im Zustand RUN?

1.6. Die Stoppuhr wird gestartet und nach 10 Sekunden wieder gestoppt. Kann man jetzt die Stoppuhr erneut starten, sodass sie ausgehend von der aktuellen Position (10 Sekunden) weiterzählt? Oder beginnt die Stoppuhr auf jeden Fall immer wieder bei null, wenn sie erneut gestartet wird? (Kurze Begründung!)

1.7. Fängt die Stoppuhr direkt an zu zählen, wenn der Start-Taster gedrückt wird? Oder läuft sie erst dann los, wenn der Start-Taster wieder losgelassen wird? (Kurze Begründung!)

1.8. Beim Zustandsübergang von TMP2 nach STOP passiert doch gar nichts (Aktion: keine). Wozu ist der Zustand TMP2 dann überhaupt erforderlich? Könnte man den Zustand TMP2 auch einfach weglassen und vom Zustand RUN direkt nach STOP wechseln, wenn der Start-Taster gedrückt wird? Würde die Stoppuhr in diesem Fall noch richtig funktionieren? (Kurze Begründung!)

1.9. In den Funktionen `count_get()` und `count_reset()` werden die beiden Funktionen `cli()` und `sei()` aufgerufen. Wozu dienen diese beiden Aufrufe?

1.10. Die Funktion `usart_line()` dient dazu, eine komplette Textzeile auf der seriellen Schnittstelle auszugeben. Woran erkennt die Funktion `usart_line()` die Länge der Textzeile (also die Anzahl der auszugebenden Zeichen)?

1.11. Wozu dient `usart_transmit(13); usart_transmit(10);` in der Funktion usart_line()?

1.12. Die beiden Taster werden aus Versehen falsch angeschlossen: Beide Taster werden statt mit GND aus Versehen mit +5V verbunden. Nun wird die Schaltung in Betrieb genommen. Woran bemerken Sie den Fehler, wie verhält sich die Stoppuhr nun?

1.13. Warum lässt sich die Stoppuhr nicht mehr richtig bedienen, wenn der Befehl `_delay_ms(10);` zu Beginn des Hauptprogramms entfernt wird?

1.14. Wie lauten die fehlenden Befehle des Hauptprogramms `main()`?

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdint.h>
#include <stdio.h>

#define BUTTON_START PB3  // Taster an PB3, interner Pullup
#define BUTTON_RESET PD7  // Taster an PD7, interner Pullup
#define LED PC0           // LED mit Vorwiderstand an PC0

// Symbolische Konstanten zur USART-Initialisierung
#define BAUD 9600
#define MYUBRR (F_CPU / 16 / BAUD - 1)

// Endlicher Automat, mögliche Zustände
#define ST_RESET 0
#define ST_TMP1  1
#define ST_RUN   2
#define ST_TMP2  3
#define ST_STOP  4

volatile uint8_t state = ST_RESET;      // Endlicher Automat, (Start-)Zustand
volatile uint16_t count_internal_ = 0;  // Aktueller Zählerstand für die Stoppuhr
uint8_t led_status = 0;                 // Aktueller LED-Zustand: 1 = an, 0 = aus

// Stoppuhr: Zählerstand abfragen
uint16_t count_get(void)
{
    uint16_t tmp = 0;
    // Siehe Aufgaben 1.9 und 1.10
    cli(); tmp = count_internal_; sei();
    return tmp;
}
```

```
// Stoppuhr: zurück auf null stellen
void count_reset(void)
{
    cli(); count_internal_ = 0; sei();  // Siehe Aufgaben 1.9 und 1.10
}

// Externe Leuchtdiode (LED) umschalten
void led_toggle(void)
{
    if(led_status)
    {
        PORTC &= ~_BV(LED);
        led_status = 0;
    }
    else
    {
        PORTC |= _BV(LED);
        led_status = 1;
    }
}

// Externe Leuchtdiode (LED) ausschalten
void led_off(void)
{
    PORTC &= ~_BV(LED);
    led_status = 0;
}

// Siehe Aufgabe 1.4
ISR(TIMER1_COMPA_vect)
{
    if(state == ST_RUN || state == ST_TMP1)
        ++count_internal_;
}

// Timer initialisieren
void init_timer(void)
{
    TCCR1B = _BV(WGM12) + _BV(CS10) + _BV(CS11);
    TIMSK1 = _BV(OCIE1A);
    OCR1A  = 24999;
    sei();
}

// USART initialisieren
void usart_init(uint16_t ubrr)
{
    UBRR0L = (uint8_t)(ubrr);
    UBRR0H = (uint8_t)(ubrr >> 8);       // Set baud rate
    UCSR0B = _BV( RXEN0) + _BV( TXEN0);  // Enable receiver & transmitter
    UCSR0C = _BV(UCSZ00) + _BV(UCSZ01);  // 8 data bits, 1 stop bit
}

// USART, einzelnes Zeichen senden
void usart_transmit(uint8_t data)
{
    while(!(UCSR0A & _BV(UDRE0))) { }    // Wait for empty transmit buffer
    UDR0 = data;                         // Put data into transmit buffer
}
```

```c
// USART, einzelnes Zeichen empfangen
uint8_t usart_receive(void)
{
    while (!(UCSR0A & _BV(RXC0))) { }    // Wait for data to be received
    return UDR0;                         // Get and return received data
}

// USART, komplette Textzeile senden, siehe Aufgabe 1.11
void usart_line(const char *text)
{
    uint16_t idx;
    for(idx = 0; text[idx] != 0; idx++)
        usart_transmit(text[idx]);

    // Siehe Aufgabe 1.12
    usart_transmit(13); usart_transmit(10);
}

// Aktuelle Stoppuhr-Zeit auf serieller Schnittstelle ausgeben
void usart_time(uint16_t count)
{
    char buf[16];
    sprintf(buf, "%6.1f s", count / 10.0);
    usart_line(buf);
}

// Ports für Taster (interne Pullups!) und LED initialisieren
void init_ports(void)
{
    // Siehe Aufgabe 1.1
}

// Wird der Start-Taster gerade gedrückt (1) oder nicht (0)?
int button_start(void)
{
    // Siehe Aufgabe 1.2
}

// Wird der Reset-Taster gerade gedrückt (1) oder nicht (0)?
int button_reset(void)
{
    // Siehe Aufgabe 1.2
}

// Hauptprogramm
int main(void)
{
    uint16_t count_old = 0;
    usart_init(MYUBRR);
    init_ports();
    init_timer();

    while(1)
    {
        uint16_t count = count_get();
        _delay_ms(10);  // Siehe Aufgabe 1.14

        // Siehe Aufgabe 1.15
```

20.14.1. TC1 Control Register A

Name: TCCR1A
Offset: 0x80
Reset: 0x00
Property: -

Aus dem Datenblatt des Mikrocontrollers ATmega328P

Bit	7	6	5	4	3	2	1	0
	COM1	COM1	COM1	COM1			WGM11	WGM10
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

Bits 4, 5, 6, 7 – COM1, COM1, COM1, COM1: Compare Output Mode for Channel
The COM1A[1:0] and COM1B[1:0] control the Output Compare pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A[1:0] bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B[1:0] bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver.

When the OC1A or OC1B is connected to the pin, the function of the COM1x[1:0] bits is dependent of the WGM1[3:0] bits setting. The table below shows the COM1x[1:0] bit functionality when the WGM1[3:0] bits are set to a Normal or a CTC mode (non-PWM).

Table 20-3. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on Compare Match.
1	0	Clear OC1A/OC1B on Compare Match (Set output to low level).
1	1	Set OC1A/OC1B on Compare Match (Set output to high level).

Bits 0, 1 – WGM10, WGM11: Waveform Generation Mode
Combined with the WGM1[3:2] bits found in the TCCR1B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. (See Modes of Operation).

Table 20-6. Waveform Generation Mode Bit Description

Mode	WGM13	WGM12 (CTC1)(1)	WGM11 (PWM11)(1)	WGM10 (PWM10)(1)	Timer/ Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

20.14.2. TC1 Control Register B

Name: TCCR1B
Offset: 0x81
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	ICNC1	ICES1		WGM13	WGM12	CS12	CS11	CS10
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

Bit 7 – ICNC1: Input Capture Noise Canceler
Writing this bit to '1' activates the Input Capture Noise Canceler. When the noise canceler is activated, the input from the Input Capture pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The Input Capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

Bit 6 – ICES1: Input Capture Edge Select
This bit selects which edge on the Input Capture pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to '1', a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the Input Capture Register (ICR1). The event will also set the Input Capture Flag (ICF1), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM1[3:0] bits located in the TCCR1A and the TCCR1B Register), the ICP1 is disconnected and consequently the Input Capture function is disabled.

Bits 3, 4 – WGM12, WGM13: Waveform Generation Mode
Refer to TCCR1A.

Bits 0, 1, 2 – CS10, CS11, CS12: Clock Select 1 [n = 0..2]
The three Clock Select bits select the clock source to be used by the Timer/Counter. Refer to Figure 20-10 and Figure 20-11.

Table 20-7. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0		1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)

20.14.12. Timer/Counter 1 Interrupt Mask Register

Name: TIMSK1
Offset: 0x6F
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
			ICIE			OCIEB	OCIEA	TOIE
Access			R/W			R/W	R/W	R/W
Reset			0			0	0	0

Bit 5 – ICIE: Input Capture Interrupt Enable
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture interrupt is enabled. The corresponding Interrupt Vector is executed when the ICF Flag, located in TIFR1, is set.

Bit 2 – OCIEB: Output Compare B Match Interrupt Enable
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector is executed when the OCFB Flag, located in TIFR1, is set.

Bit 1 – OCIEA: Output Compare A Match Interrupt Enable
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector is executed when the OCFA Flag, located in TIFR1, is set.

Aufgabe 2 von 4 (Rechnerarchitektur, ca. 10 Punkte ≙ 10 Minuten)

2.1. Erläutern Sie in einige Stichworten den Unterschied zwischen

a. einem „von-Neumann-Rechner“ und
b. einem „Harvard-Rechner“.

2.2. Was versteht man unter der „modifizierten Harvard-Architektur“ bei einem Mikrocontroller des Typs ATmega328P, der im Rahmen dieser Lehrveranstaltung verwendet wurde?

2.3. Ein Feld mit konstanten Integer-Werten ist wie folgt definiert:

```
const uint8_t my_data[] PROGMEM = { 10, 20, 30, 40, 50, 60, 70 };
uint8_t x;
```

Geben Sie einen geeigneten Befehl an, mit dem der erste Wert aus dem Feld `my_data` in die lokale Variable `x` übertragen werden kann.

2.4. Bei der Programmierung mit Interrupts müssen manche Variablen als `volatile` gekennzeichnet werden. Warum ist es nicht sinnvoll, „zur Sicherheit“ generell alle Variablen in einem Programm `volatile` zu machen?

2.5. Zum Beenden eines „normalen“ C-Programms (auf einem PC) wird die Anweisung `return 0;` ausgeführt. Bei C-Programmen auf Mikrocontrollern ist dies nicht so. Warum nicht?

Aufgabe 3 von 4: Verschiedenes (ca. 15 Punkte ≙ 15 Minuten)

3.1. Skizzieren Sie eine Schaltung zum Anschluss eines (Tast-)Schalters an den Anschluss PC1 eines ATmega328P-Mikrocontrollers. Der Schalter, der Eingang PC1 des Mikrocontrollers und der Pullup-Widerstand am/im Eingang des Mikrocontrollers sollen in Ihrer Skizze erkennbar sein. Beantworten Sie auch die folgenden beiden Fragen:

a. Warum ist der Pullup-Widerstand erforderlich?

b. Mit welchem Befehl wird der Pullup-Widerstand am Eingang PC1 aktiviert?

3.2. Der Zustand eines (Tast-)Schalters am Eingang PB0 eines ATmega328P-Mikrocontrollers soll eingelesen werden. Die folgenden Befehle funktionieren allerdings nicht. Wo liegt der Fehler?

```
// Hinweis: Die Variable "taster_gedrueckt" ist weiter oben im Programm
definiert...
if((PORTB & 0b00000001) == 0)
{
    // Taster gedrückt
    taster_gedrueckt = 1;
}
else
{
    // Taster nicht gedrückt
    taster_gedrueckt = 0;
}
```

3.3. Wie lauten die Ergebnisse der folgenden Berechnungen mit 8-Bit-Zahlen?

Ergebnisse als Binärzahl schreiben!

a) 0b00001100 | 0b00001000 = ?
b) 0b00001100 + 0b00001000 = ?
c) 0b00001100 | 0b00001100 | 0b00001100 = ?
d) 0b00001100 + 0b00001100 + 0b00001100 = ?
e) _BV(5) | _BV(5) = ?
f) _BV(5) + _BV(5) = ?
g) ~_BV(5) | _BV(5) = ?
h) ~_BV(5) & _BV(5) = ?

Aufgabe 4 von 4: Serielle Schnittstelle (ca. 10 Punkte ≙ 10 Minuten)

4.1. Wie lange dauert die Übertragung einer zip-Datei mit einer Größe von 100.000 Bytes über eine serielle Schnittstelle minimal? Die Datenübertragungsrate der Schnittstelle beträgt 19200 Bits pro Sekunde, es werden acht Daten-, ein Start- und ein Stoppbit gesendet.

4.2. Wodurch kann es bei der seriellen Datenübertragung zu einem „Overrun-Error“ kommen? Wird ein solcher Fehler durch den Sender oder den Empfänger erkannt?

4.3. Wodurch kann es bei der seriellen Datenübertragung zu einem „Framing-Error“ kommen? Nennen Sie zwei Beispiele! Wird dieser Fehler durch den Sender oder den Empfänger erkannt?

4.4. Auf der Arduino-Platine aus dem Praktikum befindet sich ein ATmega328P-Mikrocontroller. Es ist problemlos möglich, Daten über die serielle Schnittstelle des Microcontrollers an einen PC zu senden. Dies funktioniert sogar, obwohl moderne PCs über gar keine „klassischen“ seriellen Schnittstellen mehr verfügen sondern nur über einige USB-Anschlüsse.

Beschreiben Sie in wenigen Stichworten (oder mit einer passenden Skizze), wie die serielle Datenübertragung vom Mikrocontroller bis zur PC-Applikation funktioniert.

****** Viel Erfolg! ******