TECHNISCHE HOCHSCHULE LÜBECK

B.Sc. Biomedizintechnik
Fachschaft AN

PRAKTIKUM PROGRAMMIEREN VON MIKROPROZESSOREN

Dokumentation Software

Inderdeep Minhas
inderdeep.minhas@stud.th-luebeck.de

Abgabe: 14.01.2023

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Mithilfe einer Finite State Machine Model werden die Übergänge der Zustände dargestellt. Es werden zusätzlich drauf geachtet, dass die Requirements der Spezifkationen erfüllt werden.
In Abbildung 1 ist die Finite State Machine für die Barista Kaffeemaschine dargestellt.
In diesem Dokument wird die Software erläutert. Die Software wurde in der Programmiersprache C programmiert.

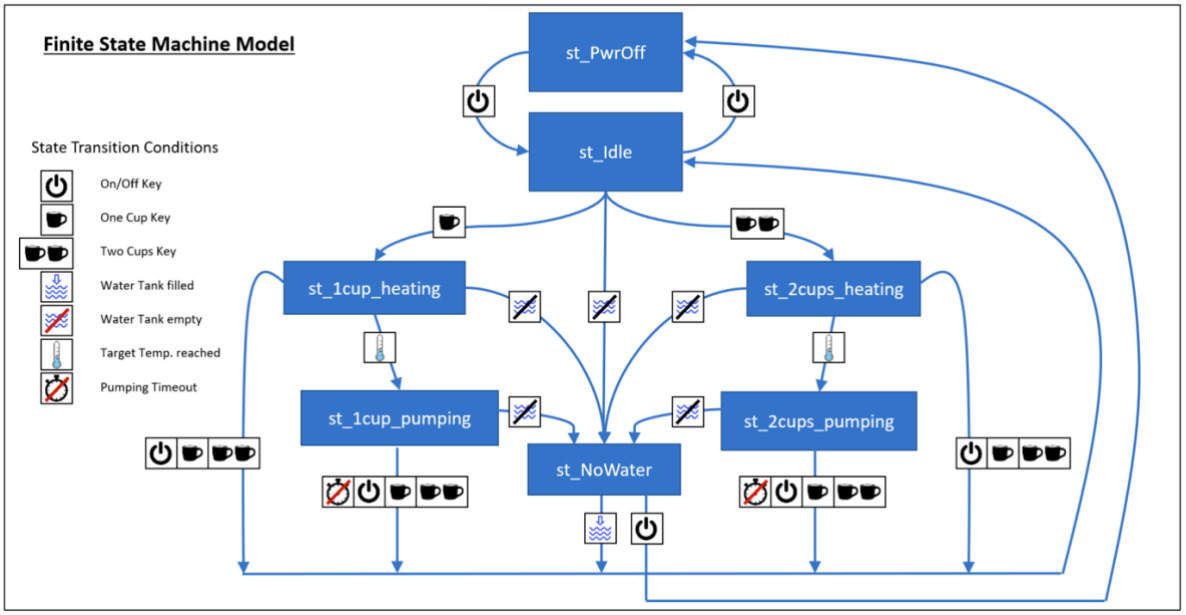

Abbildung 1 Finite State Machine

2 Erläuterungen zu den Funktionen in der Software

Im Folgenden Dokument werden die Funktionen in der Software erläutert.

Enumeration

Mithilfe einer Enumeration `state_t` werden die verschiedenen Zustände, die die Kaffeemaschine einnimmt, definiert. Die Zustände sind in der Finite State Machine in Abbildung 1 in blauen Kästchen umrahmt.
In den Requirements [1] (RQ 6) sind die Betriebszustände definiert.
Eine Enumeration weist jedem Zustand einen Wert von null aufsteigend zu.
Mithilfe des Wertes können die verschiedenen Zustände unterschieden werden.

```
enum state_t{
    st_PowerOff,
    st_Idle,
    st_one_cup_heating,
    st_one_cup_pumping,
    st_two_cup_heating,
    st_two_cup_pumping,
    st_no_water
}state;
```

Abbildung 2 Enumeration

[1] (Roloff, 2022) S.7

WaterTemp()

WaterTemp() misst die aktuelle Wassertemperatur im Heizer. Es werden zehn Werte gemessen und davon der Mittelwert gebildet.
1: steht für ungefähr 100°C und 15: steht für ungefähr 15°C. Es wird der Kanal 14 verwendet.
In Abbildung 4 sind wichtige Informationen zur Funktion.

```
uint16_t WaterTemp(void) {
    int i;
    uint16_t dat;

    ADC_ChannelConfTypeDef sConfig = {0};
    sConfig.Channel = ADC_CHANNEL_14;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);

    dat = 0 ;
    for (i=0; i<10; i++) {
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
        dat = dat + (HAL_ADC_GetValue(&hadc1) >> 8);    //lowest 8 bit ignored
    }
    return dat / 10;
}
```

Abbildung 3 WaterTemp()

Definition	`uint16_t WaterTemp(void);`
Input	void (no input)
Input values	n/a
Output	16bit integer (uint_16)
Output values	`WaterTemp()` = 15 … 1 where 1: temperature is approx. 100°C 15: temperature is approx. 15°C

Abbildung 4 WaterTemp() - Funkionserläuterung

WaterLevel_ok()

Die Funktion Waterlevel_ok gibt den Wert 0 bei geringem Wasserstand zurück ansonsten gibt es den Wert 1 zurück. Es wird der Kanal 15 verwendet. In Abbildung 6 sind wichtige Informationen zur Funktion.

```
uint16_t WaterLevel_ok(void) {
    int i;
    uint16_t dat;

    ADC_ChannelConfTypeDef sConfig = {0};
    sConfig.Channel = ADC_CHANNEL_15;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);

    dat = 0 ;
    for (i=0; i<10; i++) {
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
        dat = dat + HAL_ADC_GetValue(&hadc1);
    } // eo for
    if ( dat > 1000 ) return 1; else return 0;
} // eo of function
```

Abbildung 5 WaterLevel_ok()

Definition	`uint16_t WaterLevel_ok(void);`
Input	void (no input)
Input values	n/a
Output	16bit integer (uint_16)
Output values	`WaterLevel_ok()` = 0: contents of water tank less than minimum `WaterLevel_ok()` = 1: contents of water tank higher than minimum

Abbildung 6 WaterLevel_ok() - Funkionserläuterung

OnOffKey_pressed()

Die Funktion OnOffKey_pressed() gibt die Werte 0 und 1 zurück. Bei 0 wurde der Taster nicht gedrückt und beim Rückgabewert wird der Taster gedrückt. In Abbildung 7 ist der Code abgebildet und in Abbildung 8 sind wichtige Informationen zur Funktion.

```
uint16_t OnOffKey_pressed(void) {
    static int lastkeystate = OFF;              //is static to save state between calls

    int keystate = !HAL_GPIO_ReadPin(OnOffKey_GPIO_Port, OnOffKey_Pin); // get key state from input pin

    if ( keystate != lastkeystate ){            // change since last call?
        lastkeystate = keystate;                // save the keystate after change
        if ( keystate ) return PRESSED;         // key changed to 'pressed'
    } // eo if ( keystate != lastkeystate )

    return NOCHANGE;
}
```

Abbildung 7 OnOffKey_pressed()

Definition	`uint16_t OnOffKey_pressed(void);`
Input	void (no input)
Input values	n/a
Output	16bit integer (uint_16)
Output values	`OnOffKey_pressed()` = 0: no change of the button state (can be pressed or released)
	`OnOffKey_pressed()` = 1: the Power Button was pressed and hold since the last call of this function.

Abbildung 8 OnOffKey_pressed() - Funkionserläuterung

Cup1Key_pressed()

Die Funktion Cup1Key_pressed() gibt die Werte 0 und 1 zurück. Bei 0 wurde der Taster nicht gedrückt und beim Rückgabewert wird der Taster gedrückt. In Abbildung 9 ist der Code abgebildet und in Abbildung 10 sind wichtige Informationen zur Funktion.

```
uint16_t Cup1Key_pressed(void) {
    static int lastkeystate = OFF;              //is static to save state between calls

    int keystate = !HAL_GPIO_ReadPin(Cup1Key_GPIO_Port, Cup1Key_Pin);   // get key state from input pin

    if ( keystate != lastkeystate ){            // change since last call?
        lastkeystate = keystate;                // save the keystate after change
        if ( keystate ) return PRESSED;         // key changed to 'pressed'
    } // eo if ( keystate != lastkeystate )

    return NOCHANGE;
}
```

Abbildung 9 Cup1Key_pressed()

Definition	`uint16_t Cup1Key_pressed(void);`
Input	void (no input)
Input values	n/a
Output	16bit integer (uint_16)
Output values	**Cup1Key_pressed()** = 0: no change of the button state (can be pressed or released) **Cup1Key_pressed()** = 1: the Button for one cup of coffee was pressed and hold since the last call of this function.

Abbildung 10 Cup1Key_pressed() - Funkionserläuterung

Cup2Key_pressed()

Die Funktion Cup2Key_pressed() gibt die Werte 0 und 1 zurück. Bei 0 wurde der Taster nicht gedrückt und beim Rückgabewert wird der Taster gedrückt. In Abbildung 11 ist der Code abgebildet und in Abbildung 12 sind wichtige Informationen zur Funktion.

```
uint16_t Cup2Key_pressed(void) {
    static int lastkeystate = OFF;              //is static to save state between calls

    int keystate = !HAL_GPIO_ReadPin(Cup2Key_GPIO_Port, Cup2Key_Pin);   // get key state from input pin

    if ( keystate != lastkeystate ){            // change since last call?
        lastkeystate = keystate;                // save the keystate after change
        if ( keystate ) return PRESSED;         // key changed to 'pressed'
    } // eo if ( keystate != lastkeystate )

    return NOCHANGE;
}
```

Abbildung 11 Cup2Key_pressed()

Definition	`uint16_t Cup2Key_pressed(void);`
Input	void (no input)
Input values	n/a
Output	16bit integer (uint_16)
Output values	**Cup2Key_pressed()** = 0: no change of the button state (can be pressed or released) **Cup2Key_pressed()** = 1: the Button for two cups of coffee was pressed and hold since the last call of this function

Abbildung 12 Cup2Key_pressed() - Funkionserläuterung

LED()

Die Funktion LED() gibt keinen Wert zurück. Es wird ein Wert stat eingegeben, welches den Wert 0: OFF, 1: ON, 2: blinkt mit 1Hz und 3: blinkt mit 10Hz besitzt, In Abbildung 13 ist der Code abgebildet und in Abbildung 14 sind wichtige Informationen zur Funktion.

```
void LED(uint16_t stat) {
    static uint32_t lapcounter = 0;
    uint32_t timestamp;

    switch ( stat ) {
        case OFF: {
            HAL_GPIO_WritePin(LEDControl_GPIO_Port, LEDControl_Pin, RESET);
            break;
        } // eo case OFF:

        case ON: {
            HAL_GPIO_WritePin(LEDControl_GPIO_Port, LEDControl_Pin, SET);
            break;
        } // eo case ON:

        case SLOWBLINK: {
            timestamp = __HAL_TIM_GET_COUNTER(&htim3);
            if ( (timestamp - lapcounter) >= 4096 ) {
                HAL_GPIO_TogglePin(LEDControl_GPIO_Port, LEDControl_Pin);
                lapcounter = timestamp;
            }
            break;
        } // eo case SLOWBLINK:

        case FASTBLINK: {
            timestamp = __HAL_TIM_GET_COUNTER(&htim3);
            if ( (timestamp - lapcounter) >= 350 ) {
                HAL_GPIO_TogglePin(LEDControl_GPIO_Port, LEDControl_Pin);
                lapcounter = timestamp;
            }
            break;
        } // eo case FASTBLINK:
    } // eo switch

} // eo function
```

Abbildung 13 LED()

LED(stat)

Controls the red LED of the SENSEO machine. The LED is placed behind the power button. Non-blocking.

Definition	`void LED(uint16_t stat);`
Input	16bit integer (uint_16)
Input values	stat = 0: switches LED off stat = 1: switches LED on stat = 2: let LED blink with a frequency of appr 1Hz stat = 3: let LED blink with a frequency of appr 10Hz
Output	void (no output)
Output values	n/a

Abbildung 14 LED() - Funkionserläuterung

Pump()

Die Funktion Pump() gibt keinen Wert zurück. Es wird ein Wert stat eingegeben, welches den Wert 0: OFF, 1: ON besitzt. In Abbildung 15 ist der Code abgebildet und in Abbildung 16 sind wichtige Informationen zur Funktion.

```
void Pump(uint16_t stat) {
    if ( stat == ON ) {
        HAL_GPIO_WritePin(PumpControl_GPIO_Port, PumpControl_Pin, SET);
    }
    else {
        HAL_GPIO_WritePin(PumpControl_GPIO_Port, PumpControl_Pin, RESET);
    }
}
```

Abbildung 15 Pump()

Pump(stat)

Controls the water pump of the SENSEO machine. Non-blocking.

Definition	`void Pump(uint16_t stat);`
Input	16bit integer (uint_16)
Input values	stat = 0: switches pump off stat = 1: switches pump on
Output	void (no output)
Output values	n/a (best temperature for best coffee: 7-0)

Abbildung 16 Pump() - Funkionserläuterung

Heater()

Die Funktion `Heater()` gibt keinen Wert zurück. Es wird ein Wert stat eingegeben, welches den Wert 0: Heater OFF, 1: Heater ON besitzt. In Abbildung 17 ist der Code abgebildet und in Abbildung 18 sind wichtige Informationen zur Funktion.

```
void Heater(uint16_t stat) {
    if ( stat == ON ) {
        HAL_GPIO_WritePin(HeaterControl_GPIO_Port, HeaterControl_Pin, SET);
    }
    else {
        HAL_GPIO_WritePin(HeaterControl_GPIO_Port, HeaterControl_Pin, RESET);
    }
}
```

Abbildung 17 Heater()

Heater(stat)

Controls the water heater of the SENSEO machine. Non-blocking.

Definition	`void Heater(uint16_t stat);`
Input	16bit integer (uint_16)
Input values	stat = 0: switches heater off stat = 1: switches heater on
Output	void (no output)
Output values	n/a

Abbildung 18 Heater() - Funkionserläuterung

MX_TIM3_Init(void)

Die Funktion MX_TIM3_Init(void) deklariert den Timer TIM3. Der Prescaler ist mit 21000 definiert und der Counter-Period liegt bei 65536. LED() greift auf den Timer zu. Welches in Abbildung 13 zu sehen ist.

```
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 21000-1;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 0xFFFF;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV4;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

Abbildung 19 MX_TIM3_Init(void)

MX_TIM5_Init(void)

Die Funktion MX_TIM5_Init(void) deklariert den Timer TIM5. Der Prescaler ist mit 21000 definiert und der Counter-Period liegt bei 2^{32}. TIM5 kontrolliert die Pumpzeit der Kaffeemaschine, welches später näher erläutert wird. In Abbildung 20 ist die Einstellung von Tim5 beschrieben.

```
static void MX_TIM5_Init(void)
{

  /* USER CODE BEGIN TIM5_Init 0 */

  /* USER CODE END TIM5_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM5_Init 1 */

  /* USER CODE END TIM5_Init 1 */
  htim5.Instance = TIM5;
  htim5.Init.Prescaler = 21000-1;
  htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim5.Init.Period = 0xFFFFFFFF;
  htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim5.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim5) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim5, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim5, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

Abbildung 20 MX_TIM5_Init(void)

Watertemp_converter()

Mithilfe des Watertemp_converter() wird das Digitale Singalwert der Temperatur in Grad umgerechnet.
Der Rückgabewert ist vom Typ uint16 und ist die ungerechnete Wassertemperatur.

```
uint16_t watertemp_converter(uint16_t temp)
{
    uint16_t temp_conv = 0;
    temp_conv = -6  * temp + 105;
    return temp_conv;
}
```

Abbildung 21 Watertemp_converter()

Implementieren der Funktion

Im Folgenden werden die

Eingebundene Bibliotheken

Es wurden folgende Bibliotheken wie in Abbildung 21 eingebunden.

```
#include "main.h"

/* Private includes --
/* USER CODE BEGIN Inc
#include <stdio.h>
#include <string.h>
#include "senseo.h"
```

Abbildung 22 Bibliotheken

Die Header Datei Senseo.h besitzt Definitionen, die angewendet werden können.
In Abbildung 22 ist der Inhalt abgebildet.

```
#define PRINT_TEXT HAL_UART_Transmit(&huart2,(uint8_t*)msg,strlen(msg),HAL_MAX_DELAY)

#ifndef INC_SENSEO_H_
#define INC_SENSEO_H_

#define TIME_TO_FILL_ONE_CUP 30
#define TIME_TO_FILL_TWO_CUPS 60

#define OFF         0
#define ON          1
#define FASTBLINK   2
#define SLOWBLINK   3

#define PRESSED     1
#define RELEASED    0
#define NOCHANGE    0

#define FORCED      1
#define NOTFORCED   0


#endif /* INC_SENSEO_H_ */
```

Abbildung 23 Senseo.h

Sprintf()

In diesem Abschnitt wird mit Hilfe der Funktion `sprintf` die Werte beider Sensoren im Terminal ausgegeben.
Als erstes wird eine `BUFFER` und eine `TIMEOUT` in Abbildung 23 deklariert.
Der Buffer stellt für sprintf Speicher zu Verfügung, sodass man Text ausgelesen kann.
In Abbildung 25 ist der sprintf Befehl zu sehen. Es gibt den Zustand der Kaffeemaschine aus.
Mit `HAL_UART_Transmit` wird dafür gesorgt, dass der String im Terminal erscheint.

```
#define BUFFER_SIZE 64          // for sprintf-command
#define TIMEOUT 0xFF            // for sprintf-command
```

Abbildung 24 sprintf - 1

```
int main(void)
{
  /* USER CODE BEGIN 1 */
    char buf[BUFFER_SIZE];      // for sprintf-command
```

Abbildung 25 sprintf - 2

```
// Ausgabe Terminal
sprintf(buf,"State:%d",state);
HAL_UART_Transmit(&huart2, buf, strlen(buf), TIMEOUT);
```

Abbildung 26 sprintf -3

int main() - while ()

In der int main Funktion ist eine while Schleife deklariert. Welche unendlich oft läuft.
In der whileFunktion wird mithilfe von Switch Case Anweisungen auf die unterschiedlichen Zustände der KaM verwiesen.
In den folgenden Unterkapitel werden die verschiedenen Zustände erklärt.

st_PowerOff

Die LED, Pumpe und Heater sind ausgeschaltet.
Beim Drücken des OnOff Key wird auf st_Idle verwiesen(State Machine, Abbildung 1).

```
switch(state)
{
    // ---- OFF ----
    case st_PowerOff:
    {
        LED(0);
        Pump(0);
        Heater(0);

        if(OnOffKey_pressed() !=0)
        {
            state = st_Idle;
        }

        // Ausgabe Terminal
        sprintf(buf,"State:%d",state);
        HAL_UART_Transmit(&huart2, buf, strlen(buf), TIMEOUT);
        break;
    }
```

Abbildung 27 st_PowerOff

st_Idle

Die LED ist angeschaltet und die Pumpe und Heater sind ausgeschaltet.
Es werden durch Drücken der Tasten auf die jeweiligen Zustände wie in der State Machine in Abbildung 1 verwiesen. Durch if-Abfragen (Abbildung 27 Zeile 189 bis 202) wird kontrolliert, ob die Taste gedrückt wurde. Durch Betätigen der OnOff Taste wird der Zustand st_PowerOff erreicht. Durch Betätigen der OneCup Taste wird der Zustand st_one_cup_heating erreicht. Durch Betätigen der TwoCup Taste wird der Zustand st_two_cup_heating erreicht.
Mithilfe einer if Abfrage wird mit dem Timer TIM3 die KaM nach 5 min im Betrieb nach 5 min in den Zustand st_PowerOff gewechselt.

```
166         // ---- IDLE ----
167         case st_Idle :
168         {
169             LED(1);
170             Pump(0);
171             Heater(0);
172 
173             // start Timer 3
174             time = __HAL_TIM_GET_COUNTER(&htim3);
175 
176             // Risk: autoamtisch nach fünf Minute ausschalten
177             if ( time > 4000 ) // 1s
178             {
179                 count_auschalten ++;
180                 __HAL_TIM_SET_COUNTER(&htim3, 0);
181 
182             }
183             // Risk: autoamtisch nach fünf Minute ausschalten
184             if ( count_auschalten ==300 && time > 4000 ) // 1s * 300 = 5min
185             {
186                 state = st_PowerOff;
187             }
188 
189             if (OnOffKey_pressed() !=0)
190             {
191                 state = st_PowerOff;
192             }
193 
194             if(Cup1Key_pressed() !=0)
195             {
196                 state = st_one_cup_heating;
197             }
198 
199             if(Cup2Key_pressed() !=0)
200             {
201                 state = st_two_cup_heating;
202             }
203 
204             sprintf(buf,"State:%d",state);
205             HAL_UART_Transmit(&huart2, buf, strlen(buf), TIMEOUT);
206             break;
207 
208 
209         }
```

Abbildung 28 st_Idle

st_no_water

Die LED ist angeschaltet und die Pumpe und Heater sind in dem Zustand ausgeschaltet.
Wenn Wasserlevel ungleich null ist, also genug Wasser im Tank vorhanden ist. Schaltet der Tank auf st_Idle (State Machine, Abbildung 1).
Wenn die Taste OnOffKey gedrückt wird, schaltet sich die Kaffeemaschine aus und ist im Power Off Zustand (State Machine, Abbildung 1).
Mit der Sprintf Funktion kann nachvollzogen werden, in welchem Zustand dich die KaM befindet.

```
211             //---- NO WATER ----
212             case st_no_water :
213             {
214                 LED(1);
215                 Pump(0);
216                 Heater(0);
217
218                 if (WaterLevel_ok() != 0)
219                 {
220                     state = st_Idle;
221                 }
222
223                 if(OnOffKey_pressed() !=0)
224                 {
225                     state = st_PowerOff;
226                 }
227
228                 sprintf(buf,"State:%d",state);
229                 HAL_UART_Transmit(&huart2, buf, strlen(buf), TIMEOUT);
230                 break;
231             }
```

Abbildung 29 st_no_water

st_one_cup_heating

In diesem Zustand blinken die LED mit 1Hz.
Es wird überprüft, ob genug Wasser im Tank vorhanden ist, falls nicht wird auf den Zustand st_no_water verwiesen (Abbildung 29 Zeile 238-241).
Durch das Drücken der Taste Cup1Key, Cup2Key oder OnOffKey geht die Kaffeemaschine (Risk Mitigration) aus dem Zustand st_one_cup_heating in den Zustand st_Idle über (State Machine, Abbildung 1).
Das ist eine weitere Sicherheitsmaßnahme, die vorgenommen wurde.
Ist die Wassertemp kleiner 60 Grad wird der Heater angeschaltet. Erreicht der Heater eine Temparatur höher 70 Grad wird der Heater ausgeschaltet und geht in st_one_cup_pumping über (State Machine, Abbildung 1).
Mit der Sprintf Funktion kann nachvollzogen werden, in welchem Zustand dich die KaM befindet.

```
234             case st_one_cup_heating:
235             {
236                 LED(SLOWBLINK);
237
238                 if (WaterLevel_ok() != 1)
239                 {
240                     state = st_no_water;
241                 }
242
243                 // Risk: KaM wird durch drücken einer beliebeigen Taste auf Idle verwiesen
244                 if (OnOffKey_pressed() !=0)
245                 {
246                     state = st_Idle;
247                 }
248                 // Risk: KaM wird durch drücken einer beliebeigen Taste auf Idle verwiesen
249                 if(Cup1Key_pressed() !=0)
250                 {
251                     state = st_Idle;
252                 }
253                 // Risk: KaM wird durch drücken einer beliebeigen Taste auf Idle verwiesen
254                 if(Cup2Key_pressed() !=0)
255                 {
256                     state = st_Idle;
257                 }
258
259                 uint16_t water_temp = WaterTemp();
260
261                 if(watertemp_converter(water_temp) < 60)
262                 {
263                     Heater(1);
264                 }
265                 if(watertemp_converter(water_temp) > 70)
266                 {
267                     count = 0;
268                     Heater(0);
269                     __HAL_TIM_SET_COUNTER(&htim5, 0);
270                     state = st_one_cup_pumping;
271                 }
272
273                 sprintf(buf,"Watertemp:%d °C, State:%d\r\n",watertemp_converter(water_temp),state);
274                 HAL_UART_Transmit(&huart2, buf, strlen(buf), TIMEOUT);
275                 break;
276
277             }
```

Abbildung 30 st_one_cup_heating

st_one_cup_pumping

Es wird überprüft, ob genug Wasser im Tank vorhanden ist, falls nicht wird auf den Zustand st_no_water verwiesen (Abbildung 29 Zeile 300-303).
Durch das Drücken der Taste Cup1Key, Cup2Key oder OnOffKey geht die Kaffeemaschine (Risk Mitigration)[2] aus dem Zustand st_one_cup_heating in den Zustand st_Idle über (State Machine, Abbildung 1).
Die Zählervariable wird nach fünf Sekunden hochgezählt und der Timer auf null gesetzt (Zeile 310-314). Wenn der Counter die Zählervariable sechs erreicht, weleches sechszig Sekunden entspricht, wird zum Zustand st_Idle gewechselt, welches in der State Machine in Abbildung 1 zu sehen ist.
Mit der Sprintf Funktion kann nachvollzogen werden, in welchem Zustand dich die KaM befindet.

```
277             }
278             // ---- ONE CUP PUMPING ----
279             case st_one_cup_pumping:
280             {
281
282                 // Risk: KaM wird durch drücken einer beliebeigen Taste auf Idle verwiesen
283                 if (OnOffKey_pressed() !=0)
284                 {
285                     state = st_Idle;
286                 }
287                 // Risk: KaM wird durch drücken einer beliebeigen Taste auf Idle verwiesen
288                 if(Cup1Key_pressed() !=0)
289                 {
290                     state = st_Idle;
291                 }
292                 // Risk: KaM wird durch drücken einer beliebeigen Taste auf Idle verwiesen
293                 if(Cup2Key_pressed() !=0)
294                 {
295                     state = st_Idle;
296                 }
297
298                 LED(SLOWBLINK);
299
300                 if (WaterLevel_ok() != 1)
301                 {
302                     state = st_no_water;
303                 }
304                 if (WaterLevel_ok() != 0)
305                 {
306
307                     Pump(1);
308                     time = __HAL_TIM_GET_COUNTER(&htim5);
309
310                     if ( time > 20000 ) // 5s
311                     {
312                         count ++;
313                         __HAL_TIM_SET_COUNTER(&htim5, 0);
314
315                     }
316
317                     if ( count== 6 && time > 20000 ) // 5s * 6 = 30s
318                     {
319                         state = st_Idle;
320                     }
321                 }
322                 sprintf(buf,"Time: %d State:%d\r\n",time,state);
323                 HAL_UART_Transmit(&huart2, buf, strlen(buf), TIMEOUT);
324                 break;
325             }
```

Abbildung 31 st_one_cup_pumping

[2] (Minhas, 2023)

st_two_cup_heating

In diesem Zustand blinken die LED mit 10Hz.
Es wird überprüft, ob genug Wasser im Tank vorhanden ist, falls nicht wird auf den Zustand st_no_water verwiesen
Durch das Drücken der Taste Cup1Key, Cup2Key oder OnOffKey geht die Kaffeemaschine aus dem Zustand st_one_cup_heating in den Zustand st_Idle über(State Machine, Abbildung 1). Das ist eine weitere Sicherheitsmaßnahme, die vorgenommen wurde.
Ist die Wassertemp kleiner 60 Grad wird der Heater angeschaltet. Erreicht der Heater eine Temparatur höher 70 Grad wird der Heater ausgeschaltet und geht in st_one_cup_pumping über (State Machine, Abbildung 1).
Mit der Sprintf Funktion kann nachvollzogen werden, in welchem Zustand dich die KaM befindet.

```
// ---- TWO CUP HEATING ----
case st_two_cup_heating:
{
    LED(FASTBLINK);

    if (WaterLevel_ok() != 1)
    {
        state = st_no_water;
    }

    uint16_t water_temp = WaterTemp();

    if(watertemp_converter(water_temp) < 70)
    {
        Heater(1);
    }
    if(watertemp_converter(water_temp) > 80)
    {
        count = 0; // set counter to 0
        Heater(0);
        __HAL_TIM_SET_COUNTER(&htim5, 0); // set timer to 0
        state = st_two_cup_pumping;
    }

    sprintf(buf,"Watertemp:%d °C, State:%d\r\n",watertemp_converter(water_temp),state);
    HAL_UART_Transmit(&huart2, buf, strlen(buf), TIMEOUT);
    break;
}
```

Abbildung 32 st_two_cup_heating

st_two_cup_pumping

Es wird überprüft, ob genug Wasser im Tank vorhanden ist, falls nicht wird auf den Zustand st_no_water verwiesen
Durch das Drücken der Taste Cup1Key, Cup2Key oder OnOffKey geht die Kaffeemaschine (Risk Mitigration)[3] aus dem Zustand st_one_cup_heating in den Zustand st_Idle über (State Machine, Abbildung 1).
Die Zählervariable wird nach fünf Sekunden hochgezählt und der Timer auf null gesetzt. (Minhas, 2023)Wenn der Counter die Zählervariable sechs erreicht, weleches sechszig Sekunden entspricht, wird zum Zustand st_Idle gewechselt, welches in der State Machine in Abbildung 1 zu sehen ist.
Mit der Sprintf Funktion kann nachvollzogen werden, in welchem Zustand dich die KaM befindet.

```
case st_two_cup_pumping:
{
    LED(FASTBLINK);

    if (WaterLevel_ok() != 1)
    {
        state = st_no_water;
    }
    if (WaterLevel_ok() != 0)
    {

        Pump(1);
        time = __HAL_TIM_GET_COUNTER(&htim5);

        if ( time > 20000 ) // 5s
        {
            count ++;
            __HAL_TIM_SET_COUNTER(&htim5, 0);

        }

        if ( count== 12 && time > 20000 ) // 5s * 12 = 60s
        {
            state = st_Idle;
        }
    }
    sprintf(buf,"Time: %d State:%d\r\n",time,state);
    HAL_UART_Transmit(&huart2, buf, strlen(buf), TIMEOUT);
    break;
}
```

Abbildung 33 st_two_cup_pumping

[3] (Minhas, 2023)

Dateien und IDE

Alle Dateien die für die Erstellung der Software zu finden sind, sind in der Datei SW_BARISTA.zip zu finden, einschließlich dieses Dokument.
Es wird mit der Software DIE STM32CubeIDE Version 1.10.1 verwendet.
In der SW_BARISTA.zip ist die senseo.h und main.c zu finden.

Literaturverzeichnis

Minhas, I. (14. 01 2023). Risk_Mitigration. Hamburg.
Roloff, H. (01. 12 2022). Das BARISTA_Projekt - Anforderungen. Hamburg.