Temperatur- & Luftfeuchtigkeit-Anzeige

In unserem Projekt haben wir die Temperatur sowie die Luftfeuchtigkeit von einem MEMS Shield gelesen und die Daten über eine UART an ein Terminal und einen anderen Mikrocontroller (µC) geschickt, auf dem diese mit einer 4 stelligen 7-Segment-Anzeige ausgegeben wurden.

Kalibrierung

Um die Messdaten des MEMS Shields in nutzbare Einheiten umrechnen zu können, mussten wir zuerst die Referenzwerte auslesen und berechnen, in welchem Verhältnis die Messwerte zu den Einheiten Grad Celsius und prozentualer Luftfeuchtigkeit stehen.

```
175 void CalibrateSensor() {
176     // Enable the sensors and set the measurement frequency to 1 Hz
177     HAL_I2C_Master_Transmit(&hi2c1, HT221_write, &sensor_on, 2, 1000);
178
179     // Get degree
180     //T0 low
181     RxTx_I2C(&T0_deg_ad_l, &T0_deg_l);
182     // T1 low
183     RxTx_I2C(&T1_deg_ad_l, &T1_deg_l);
184     // deg_h
185     RxTx_I2C(&deg_ad_h, &deg_h);
186     T0_deg_h = deg_h & 0x03;
187     T1_deg_h = deg_h & 0x0C;
188
189     T0_deg = ((T0_deg_h << 8) | T0_deg_l) / 8;
190     T1_deg = ((T1_deg_h << 6) | T1_deg_l) / 8;
191
192     // Get T0 and T1 Value
193     RxTx_I2C(&T0_val_ad_l, &T0_val_l);
194     RxTx_I2C(&T0_val_ad_h, &T0_val_h);
195     RxTx_I2C(&T1_val_ad_l, &T1_val_l);
196     RxTx_I2C(&T1_val_ad_h, &T1_val_h);
197
198     T0_val = (T0_val_h << 8) | T0_deg_l;
199     T1_val = (T1_val_h << 8) | T1_deg_l;
200
201     // Calculate the conversion ratio
202     counts_per_deg = (T1_val - T0_val) / (T1_deg - T0_deg);
203
204     // ------------------------------------------------------------------
205     // Get deg
206     //T0 low
207     RxTx_I2C(&H0_deg_ad_l, &H0_deg_l);
208     // T1 low
209
210     RxTx_I2C(&H1_deg_ad_l, &H1_deg_l);
211     // deg_h
212     H0_deg = (H0_deg_l) / 2;
213     H1_deg = (H1_deg_l) / 2;
214
215     // Get T0 and T1 Value
216     RxTx_I2C(&H0_val_ad_l, &H0_val_l);
217     RxTx_I2C(&H0_val_ad_h, &H0_val_h);
218     RxTx_I2C(&H1_val_ad_l, &H1_val_l);
219     RxTx_I2C(&H1_val_ad_h, &H1_val_h);
220     H0_val = (H0_val_h << 8) | H0_deg_l;
221     H1_val = (H1_val_h << 8) | H1_deg_l;
222
223     // Calculate the conversion ratio
224     H_counts_per_deg = (H1_val - H0_val) / (H1_deg - H0_deg);
```

Abbildung 1: CalibrateSensor Funktion

Dabei wird oft per I2C auf die Register des HT221 zugegriffen. Deswegen haben wir diese Kommunikation (Setzen des Pointers auf ein Register und Lesen des Inhalts) in eine eigene Funktion ausgelagert.

```
162 /*
163  * Get Value from HT221 at addr into store
164  */
165 void RxTx_I2C(uint8_t* addr, uint8_t* store) {
166     // Set internal HT221 pointer to the required address
167     HAL_I2C_Master_Transmit(&hi2c1, HT221_write, addr, 1, 1000);
168     // Receive value
169     HAL_I2C_Master_Receive(&hi2c1, HT221_read, store, 1, 1000);
170 }
```

Abbildung 2: RxTx_I2C Funktion

Daten messen und senden

Um die Daten an den zweiten Mikrocontroller zu schicken, haben wir bei beiden Controllern die UART2-Schnittstelle verwendet. Dafür wurden die Pins PD5 (TX) und PD6 (RX) gekreuzt miteinander verbunden sowie beide Controller mit einem weiteren Kabel auf das gleiche Massepotential gebracht.

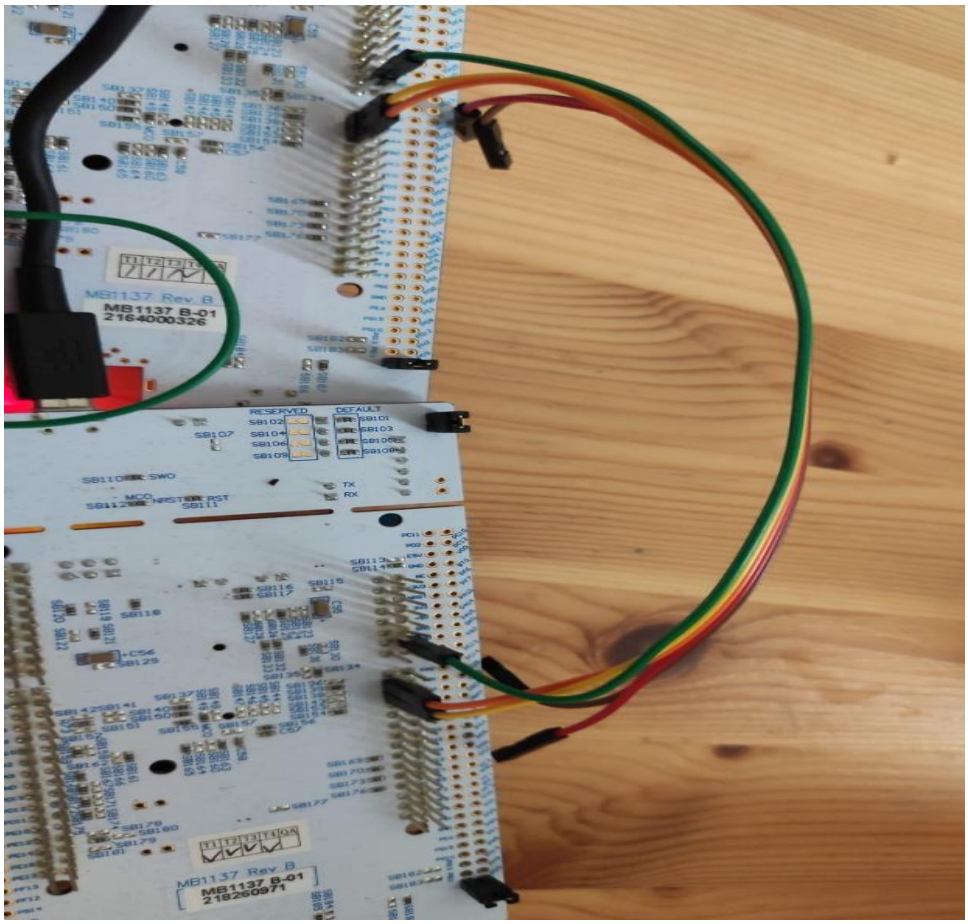


Abbildung 3: Verbindung der beiden Controller

Für das Senden an den PC wird alle 5 Sekunden die UART3-Schnittstelle aktiviert, die auf dem Board über den ST-Link mit dem PC verbunden ist.

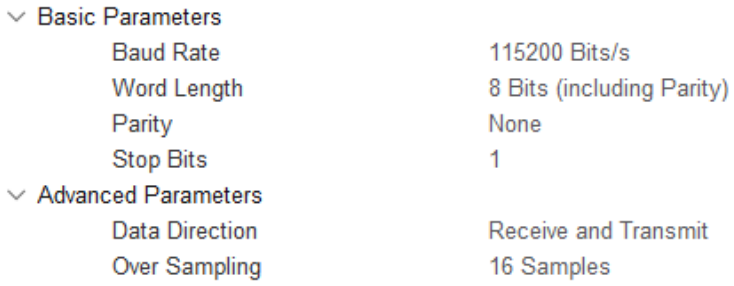


Abbildung 4: Konfuguration einer UART Schnittstelle

Das Starten der Sendevorgänge ist über zwei Basic Timer implementiert. Der Timer 6 triggert alle 5 Sekunden das Senden an den PC und der Timer 7 jede Sekunde die Messung am HT221 und das Senden an den zweiten µC.

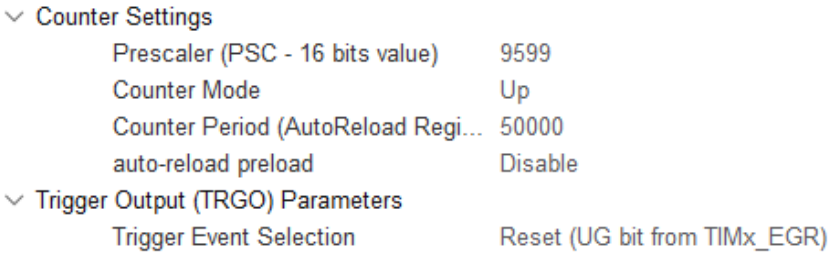


Abbildung 5: Konfiguration des Timer 6

Dabei müssen die Interrupt-Requests der beiden Timer die gleiche NVIC-Preemption-Priority-Gruppe haben, damit nicht eine ISR die andere unterbricht und so die Übertragung gestört wird.

Wie in Abbildung 6 zu sehen ist, werden die Messwerte vor dem Senden erst in ihre jeweiligen Einheiten umgerechnet.

```
598 /* USER CODE BEGIN 4 */
599 HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef * htim)
600 {
601     // Check if ISR is called by timer 6 (every 5 seconds) or by timer 7 (every second)
602     if(htim == &htim6){
603         // Print in Terminal
604         sprintf(msg, "Measurement: \r \n \n");
605         HAL_UART_Transmit(&huart3, &msg, strlen(msg), 1000);
606         sprintf(msg, "Temperature: %d \r \n \n", tempInC);
607         HAL_UART_Transmit(&huart3, &msg, strlen(msg), 1000);
608         sprintf(msg, "Humidity: %d \r \n \n \n", humInC);
609         HAL_UART_Transmit(&huart3, &msg, strlen(msg), 1000);
610     }
611     else {
612         // Measure temperature
613         RxTx_I2C(&tempLowAd, &tempLow);
614         RxTx_I2C(&tempHighAd, &tempHigh);
615         temp = (tempHigh << 8) | tempLow;
616         tempInC = ((temp - T0_val) / counts_per_deg) + T0_deg;
617 
618         // Measure humility
619         RxTx_I2C(&humLowAd, &humLow);
620         RxTx_I2C(&humHighAd, &humHigh);
621         hum = (humHigh << 8) | humLow;
622         humInC = ((hum - H0_val) / H_counts_per_deg) + H0_deg;
623 
624         // Send with UART to other microcontroller
625         sprintf(msg, "t%3dh%3d", tempInC, humInC); //'txxxhxxx'
626         HAL_UART_Transmit(&huart2, &msg, strlen(msg), 1000);
627     }
628 
```

Abbildung 6: Messen & Senden von Temperatur und Luftfeuchtigkeit

Daten empfangen

Auf dem zweiten Mikrocontroller werden die Messdaten von der UART2 empfangen und mit jeder vollständigen Datenübertragung per DMA in ein char Array kopiert. Hierzu erfolgt die Übertragung immer in dem Format „t-12h 34“, also immer Temperatur und Luftfeuchtigkeit hintereinander, sodass sie per DMA an ein Feld im Speicher abgelegt werden können und über die Startadresse in diesem Feld Temperatur oder Luftfeuchtigkeit angezeigt werden kann.

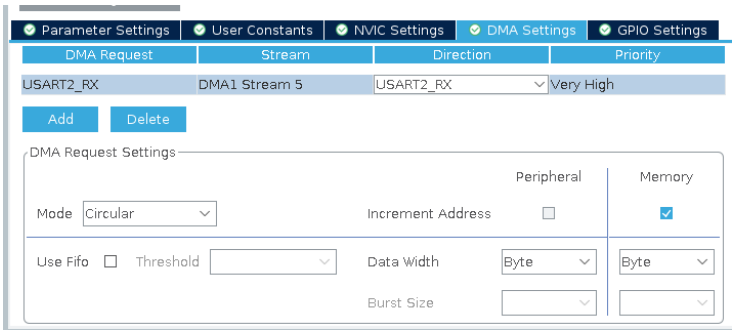


Abbildung 7: Konfiguration des UART-DMA

In der main-Routine muss dann nur noch einmalig die UART aktiviert werden, den Rest übernimmt die DMA-Einheit.

```
178
179   HAL_TIM_Base_Start_IT(&htim6);            // Start timer 6 for switching the digits
180   HAL_UART_Receive_DMA(&huart2, &data, 8);  // Start listening for incoming UART transa
181
182
183   /* USER CODE END 2 */
184
185   /* Infinite loop */
186   /* USER CODE BEGIN WHILE */
187   while (1)
188   {
189     /* USER CODE END WHILE */
190
191     /* USER CODE BEGIN 3 */
192   }
193   /* USER CODE END 3 */
194 }
```

Abbildung 8: main-Routine

7 Segment Anzeige

Die 7-Segment-Anzeige mit 4 Ziffern auf dem Multi-Function-Shield wird über zwei Schieberegister angesteuert, die miteinander verkettet sind. Zur Anzeige der Daten müssen daher sowohl die ausgewählte Ziffer als auch das anzuzeigende Zeichen per Serial-Peripheral-Interface (SPI) durch die Schieberegister geschoben werden, bevor sie per Latch-Signal ausgegeben und damit angezeigt werden können. Um für jede Ziffer ein anderes Zeichen anzuzeigen, müssen die 4 Ziffern nacheinander angesteuert werden. Hierfür generiert der Basic Timer 6 alle 3ms einen Interrupt-Request.

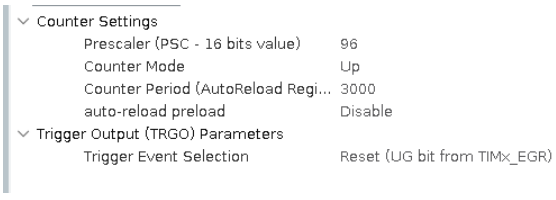


Abbildung 9: Konfiguration des Timer 6

In dessen ISR werden die Daten in das entsprechende Bitmuster umgerechnet, welches anschließend per SPI an die Schieberegister übertragen wird, bevor im nächsten Durchlauf mit der nächsten Ziffer fortgefahren wird. Die SPI-Einheit auf dem µC löst bei abgeschlossener Übertragung wiederum einen Interrupt aus, der am Latch-Pin eine steigende Flanke und damit das Signal zur Anzeige erzeugt.

```c
 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    // If triggered by timer 6 (ca. 300Hz): Send the values for the current digit per SPI to the s
    if (htim == &htim6)
    {
        // Reload the output field with the right data (temperature: data[0]; humidity: data[4])
        reloadNumber(&data[isHum * 4]);
        HAL_SPI_Transmit_IT(&hspi1, &number[currentDigit], 1);
        currentDigit = (currentDigit + 1) % 4;
    }
    // If triggered by timer 7 (every 5s if enabled): Toggle between temperature and humidity and
    else if (htim == &htim7)
    {
        isHum ^= 1;
        reloadNumber(&data[(isHum) * 4]);
    }
}

/*
 * ISR called when SPI transmission completed
 */
void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi)
{
    // Generate a rising edge to the shift registers' latch pin to output the transmitted values
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_14, RESET);
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_14, SET);
}
```

Abbildung 10: ISRs für Timer und SPI

Umrechnung der Daten

Die Generierung des Bitmusters für die Schieberegister übernimmt die Funktion „ReloadNumber“. Sie erstellt für jede der vier Ziffern ein Bitmuster, welches dann per SPI gesendet werden kann.

```c
 */
void reloadNumber(char *display)
{
    // Checking if t or h should be displayed as first character
    if (display[0] == 't')
    {
        number[0] = switch_digits[0] << 8 | digits[12] << 1 | 1;    //
    }
    else
    {
        number[0] = switch_digits[0] << 8 | digits[13] << 1 | 1;
    }

    // Checking if the value is negative (show '-'), positive or has three digits for the second digit
    if (display[1] == ' ')
    {
        number[1] = switch_digits[1] << 8 | digits[10] << 1 | 1;

    }
    else if (display[1] == '-')
    {
        number[1] = switch_digits[1] << 8 | digits[11] << 1 | 1;
    }
    else
    {
        number[1] = switch_digits[1] << 8 | digits[display[1] - 0x30] << 1 | 1;
    }

    // Setting the last two digits
    number[2] =  switch_digits[2] << 8 | digits[display[2] - 0x30] << 1 | 1;
    number[3] =  switch_digits[3] << 8 | digits[display[3] - 0x30] << 1 | 1;
}
/* USER CODE END 0 */
```

Abbildung 11: reloadNumber Funktion

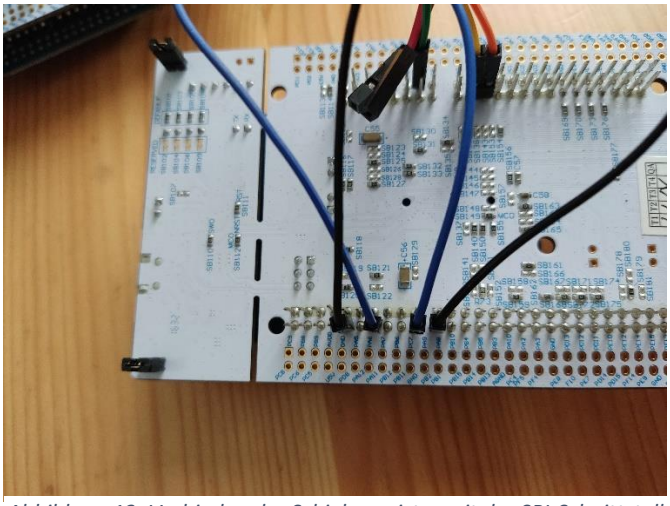
Dazu wird sowohl der Index der Ziffer als auch das entsprechende Zeichen mit jeweils einer Bitmap aufgelöst und die resultierenden Bitmuster aneinandergehängt.

```
53
54  /* USER CODE BEGIN PV */
55  uint8_t currentDigit = 0;    // Currently activated digit
56  uint8_t isHum = 0;           // Flag if the display shows temperture or hum
57  char data[8];                // Field containing the data received from UAR
58                               // Data is send in the format: [t-32h 67] (one
59
60  // Field with 7 Bits (low active) for displaying a character as 7 segment
61  uint8_t digits[] = {
62          0b0000001, // 0
63          0b1001111, // 1
64          0b0010010, // 2
65          0b0000110, // 3
66          0b1001100, // 4
67          0b0100100, // 5
68          0b0100000, // 6
69          0b0001111, // 7
70          0b0000000, // 8
71          0b0000100, // 9
72          0b1111111, // 10: Nothing (space)
73          0b1111110, // 11: minus
74          0b1110000, // 12: t
75          0b1101000, // 13: h
76  };
77
78  // Field for addressing the several digits through the shift register
79  uint8_t switch_digits[] = {
80          0b10000000,
81          0b01000000,
82          0b00100000,
83          0b00010000,
84  };
```

Abbildung 12: Bitmaps

Anschluss des Multi-Function-Shields

Da beim Aufstecken des Multi-Function-Shields die Anschlüsse der 7-Segment-Anzeige nicht direkt mit Pins einer SPI-Einheit des Mikrocontrollers verbunden werden, mussten die entsprechenden Leitungen wie folgt umgelegt werden:



PF12 → PA7

PF13 → PA5

Abbildung 13: Verbinden der Schieberegister mit der SPI-Schnittstelle

Auswahl von Temperatur und Luftfeuchtigkeit

Zur Auswahl zwischen Temperatur, Luftfeuchtigkeit oder periodischem Wechsel durch die Buttons wurden deren Pins so konfiguriert, dass bei Knopfdruck eine Flanke über den EXTI einen Interrupt triggert. In dessen ISR wird je nach Pin ein Flag gesetzt, das zeigt, ob aktuell Temperatur oder Luftfeuchtigkeit angezeigt werden soll.

```
 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    // Button left: show temperature
    if (GPIO_Pin == GPIO_PIN_0)
    {
        // Stop timer 7 to disable automatically switching between temperature and humidity
        HAL_TIM_Base_Stop_IT(&htim7);
        isHum = 0;
    }
    // Button right: show humidity
    else if (GPIO_Pin == GPIO_PIN_1)
    {
        HAL_TIM_Base_Stop_IT(&htim7);
        isHum = 1;
    }
    // Button center: Enable automatically switching between temperature and humidity
    else if (GPIO_Pin == GPIO_PIN_3)
    {
        // Resetting the timer's counter register so it start always with 5s left
        __HAL_TIM_SET_COUNTER(&htim7, 0);
        // Starting the timer 7 which switches every 5s
        HAL_TIM_Base_Start_IT(&htim7);
        // Toggle flag and reload output to switch display directly after pressing the button
        isHum ^= 1;
        reloadNumber(&data[isHum * 4]);
    }
}

```

Abbildung 15: EXTI ISR

Zum periodischen Wechsel zwischen beiden wird an dieser Stelle für den entsprechenden Knopf der Timer 7 gestartet, der alle 5 Sekunden über einen Interrupt automatisch das oben genannte Flag toggled. Um diese Funktion zu deaktivieren, muss nun auch bei den anderen beiden Buttons der Timer 7 wieder gestoppt werden (und dessen Counter Register vor einem Neustart zurückgesetzt werden).

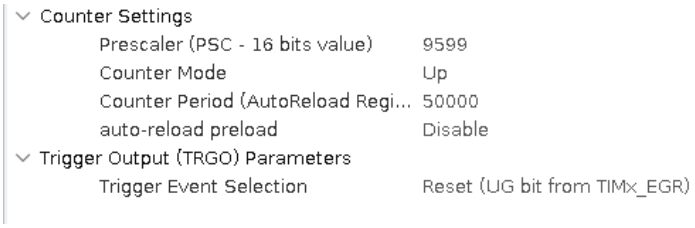


Abbildung 14: Konfiguration des Timer 7