JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

IoT Visualisierung erweitert

Systemtechnik Labor
5BHIT 2017/18

Martin Wölfer

Version 0.1
Begonnen am 24.Januar.2018
Beendet am 25. Januar 2018

Note:
Betreuer: WEIJ

Inhaltsverzeichnis

1 Aufgabenstellung

Zurück zu: Übungen EK: Emb... Im Abschnitt "Übungen EK: Embedded Devices - Weiser"gibt es im Ordner ”Unterlagen zu den Übungen” ein Dokument namens ”Datenvisualisierung.pdf”. Darin ist eine Aufgabe für die Grundkompetenzen als auch eine Aufgabe für die erweiterten Kompetenzen beschrieben. Führe die Übungen der zweiten Aufgabe (erweiterte Kompetenzen) hier durch und gib dann das entsprechende Protokoll und den Code hier ab.

Es gibt auchim selben Ordner noch ein entsprechendes zip-File mit Java-Klassen, welche du benötigst. Weiters benötigst du auchdas Ergebnis der ersten Aufgabe zm Thema IoT-Visualisierung.

 Martin Wölfer

2 Aufbau

Der USART Adapter muss zusätzlich mit dem Microboard an einem USB-Port hängen. Der Adapter steckt dann mit dem **weißen** Port an **PA9** und **gelb** an **PA10**:

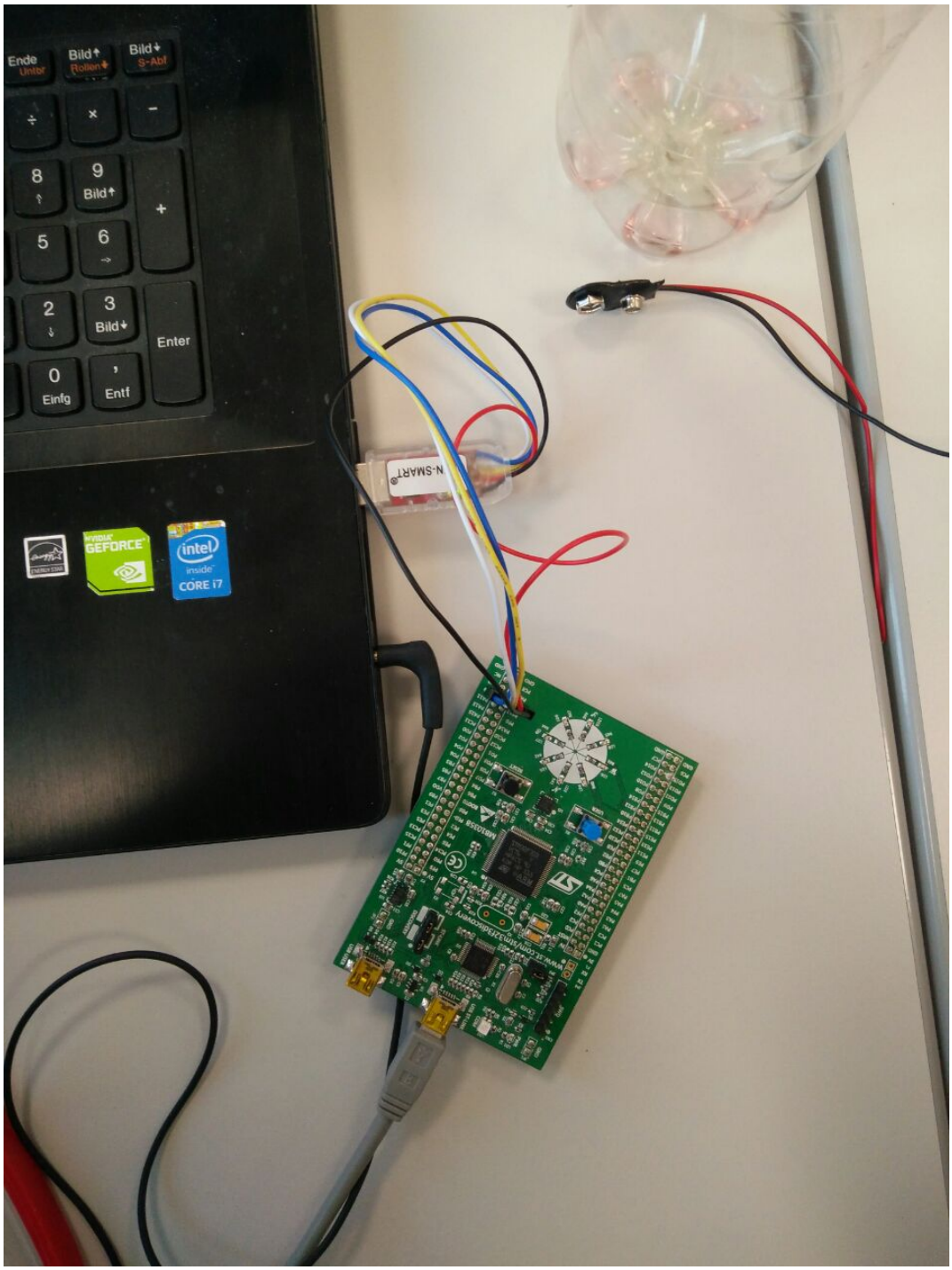

Abbildung 1: Aufbau mit dem Adapter und dem Microboard

3 Daten generieren auf dem Board

Der nächste Schritt war es nun, ein Programm in C zu schreiben, welches die Daten, welche in der Grundkompetenz in `DataSimulator.java` simuliert wurden, vom Board generieren zu lassen.

3.1 Projekt erstellen

Dazu wurde zuerst ein neues Projekt in der workbench angelegt, wobei zu beachten ist, dass in der neueren Version der workbench die HAL Library nun in jedem Projekt einzeln anzulegen ist und es nicht mehr möglich ist sie als einzelnes Projekt im Workspace liegen zu haben.

3.2 Konstanten

Die Parameter welche beim `DataSimulator.java` beim initialisieren angegeben wurden, werden hier nun als globale Konstanten angelegt:

```
double initialFrequency = 1;
double deltaFrequency = 0.015;
double valuesPerTimeUnit = 100;
double maxFrequency = 5;
double minFrequency = 0.5;
```

3.3 Port für USART konfigurieren

Der nächste Schritt war es nun, den Port bzw. die einzelnen Pins für den USART zu initialisieren. Wie schon im Aufbau erwähnt, wurde sich hier für die Pins **10** und **9** am Port **A** entschieden:

```
void USART1_GPIO_Configuration(void) {
    GPIO_InitTypeDef GPIO_InitStruct;
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_9 | GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

3.4 USART konfigurieren

Dafür wurde neben den Konstanten auch noch eine globale Variable angelegt, namens `UartHandle` vom Typen `UART_HandleTypeDef`.

Als Baudrate, also die Rate wie schnell die Daten ”refreshed” werden, habe ich das 4. Fache der seriellen Schnittstelle des USB Adapters ausgewählt. Dies konnte ausgelesen werden mit:

```
stty -F /dev/ttyUSB0
```

Und lieferte Folgendes Ergebnis

```
speed 9600 baud; line = 0;
-brkint -imaxbel
```

Also wurde für die Baudrate **38400** verwendet, um eine flüssigere Animation zu gestalten:

```
void USART1_Configuration(void) {
    USART1_GPIO_Configuration();
    __HAL_RCC_USART1_CLK_ENABLE();
    UartHandle.Instance = USART1;
    UartHandle.Init.BaudRate = 38400;
    UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
    UartHandle.Init.StopBits = UART_STOPBITS_1;
    UartHandle.Init.Parity = UART_PARITY_NONE;
    UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    UartHandle.Init.Mode = UART_MODE_TX_RX;
    HAL_UART_Init(&UartHandle);
}
```

3.5 Main Funktion schreiben

Zuerst werden alle Konfigurationsfunktionen aufgerufen:

```
USART1_GPIO_Configuration();
HAL_Init();
USART1_Configuration();
```

Und danach werden alle Variablen für die Datengenerierung initialisiert:

```
char str1[100];
double actualFrequency = initialFrequency;
double actDeltaFrequency = deltaFrequency;
int values[100];
double waveTime = 0;
```

Danach beginnt die while-True Schleife, um die Daten stetig an das Board zu senden:

```
for(;;) {
    // Zuerst wird 100 mal iteriert, weil jede Sinus Kurve 100 Werte haben soll
    for (int i=0;i<100;i++) {
        // Es werden die Werte generiert fuer die momentane Frequenz und WaveTime
        values[i] = (int)(10000*sin(waveTime));
        // WaveTime wird angepasst
        waveTime = waveTime + 2*M_PI*actualFrequency/100;
        if (waveTime > 2*M_PI) {
        waveTime = waveTime - 2*M_PI;
        }
    }
    // Frequenzen wird angepasst
    actualFrequency = actualFrequency * (1+actDeltaFrequency);
    if (actualFrequency>5) {
        actDeltaFrequency = -0.015;
    }
    if (actualFrequency<minFrequency) {
        actDeltaFrequency = 0.015;
    }

    // Daten werden an UART weitergeleitet
    for (int var = 0; var < 100; ++var) {
        if(var == 0){
            sprintf(str1, "$%d", values[var]);
            HAL_UART_Transmit(&UartHandle, (uint8_t *)str1, strlen(str1), 5000);
        }else if(var == 99){
            sprintf(str1, ",%d\n", values[var]);
            HAL_UART_Transmit(&UartHandle, (uint8_t *)str1, strlen(str1), 5000);
        }else{
            sprintf(str1, ",%d", values[var]);
            HAL_UART_Transmit(&UartHandle, (uint8_t *)str1, strlen(str1), 5000);
        }
    }
}
```

4 Daten auslesen vom dem Board

Anstatt die Daten nun in `DataSimulator.java` zu simulieren, werden sie in `DataGenerator.java` vom Board ausgelesen.

4.1 Attribute

Zuerst werden die nötigen Attribute angelegt:

```
static SerialPort sp;
static ArrayList<String> data = new ArrayList<String>();
static DataBuffer db;
```

Als nächstes werden im Konstruktor alle relevanten Objekte und Variablen initialisiert, und der Parameter übernommen:

```
public DataGenerator(DataBuffer db) {
    this.db = db;

    // Da ich in Linux arbeite, musste ich zuerst den Portnamen ausgeben, damit ich erkenne wie dieser
        heisst
    System.out.println(SerialPortList.getPortNames()[0]);
    // Das Ergebnis war /dev/ttyUSB0
    this.sp = new SerialPort("/dev/ttyUSB0");

    try {
        // Hilfe von Johannes Bishara
        this.sp.openPort();
        this.sp.setParams(38400, 8, 1, 0);
        int mask = SerialPort.MASK_RXCHAR + SerialPort.MASK_CTS + SerialPort.MASK_DSR;
        this.sp.setEventsMask(mask);
        this.sp.addEventListener(new SerialPortReader());
    } catch (SerialPortException ex) {
        // Bei einer Exception einen Fehler ausgeben
        System.out.println(ex);
}
```

4.2 SerialPortEventListener implementieren

Als nächstes wird der SerialPortEventListener implementiert:

```
// Methode wird ueberladen
public void serialEvent(SerialPortEvent event) {
try {
String getdata = sp.readString(event.getEventValue());
String[] parts = getdata.split("\\$");

if (data.size() > 0)
data.set(data.size() - 1, data.get(data.size() - 1) + parts[0]);
for (int i = 1; i < parts.length; i++) {
data.add(parts[i]);
}

String daten = "";
try {
daten = data.get(data.size() - 1);
} catch (ArrayIndexOutOfBoundsException e) {
return;
}

// Aus der Datenliste werden die einzelnen Teile extrahiert mit split
String[] teile = daten.split(",");
```

```
23 // Falls es 100 sind, was vorgegebn ist durch den Generator am Board
   if (teile.length == 100) {
25 // Die Liste welche dann an datenbuffer uebergeben wird
   int fertig[] = new int[100];
27 for (int i = 0; i < fertig.length; i++) {
   // Falls ein Teil nicht null ist
29 if (teile[i] != null) {
   // Versuchen den String zu einem Int zu parsen
31 int d = 0;
   try {
33 d = Integer.parseInt(teile[i].trim());
   } catch (NumberFormatException e) {
35 // do nothing, encoding error
   }
37 fertig[i] = d;
   }
39 }
   db.addList(fertig);
41 }
   } catch (SerialPortException ex) {
43 ex.printStackTrace();
   }
45 }
```

5 Ergebnis

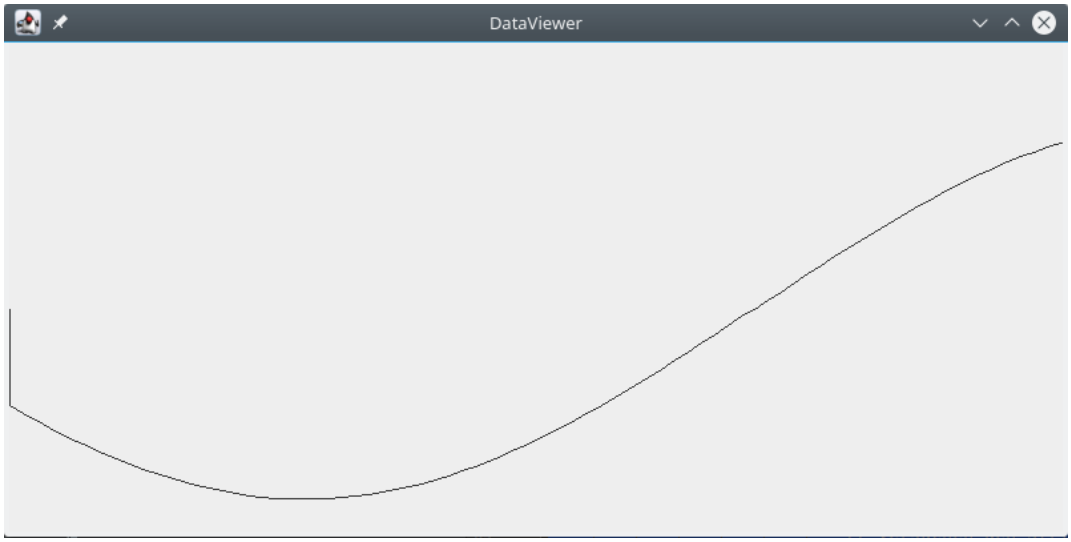

Abbildung 2: Ergebnis

Abbildungsverzeichnis