Testat- Aufgabe zum OOSE-Labor

Es wird von Ihnen erwartet, dass Sie die Übungsaufgaben des OOSE-Labors zuvor erfolgreich bearbeitet haben, um das dort Gelernte hier nun umsetzen zu können.

Aufgabe: Realzeituhr mit Weckfunktion, Temperatur- und Luftfeuchtemessung sowie Temperaturwarnfunktion

Im Rahmen der Testat-Aufgabe sollen Sie eine Realzeit-Uhr mit Weckfunktion sowie Temperatur- und Luftfeuchteanzeige entwerfen und implementieren. Es stehen Ihnen alle Software-Module aus Runtime zur Verfügung. Neu für Sie ist das Modul ADConverter zum Auslesen von analogen Sensoren sowie die notwendigen Sensoren. Mittels Drehgeber werden Einstellungen komfortabel vorgenommen; dafür steht das QuadEncoder Modul zur Verfügung.

Zur Temperatur- und Feuchtemessung bekommen Sie analoge Sensoren, die über A/D-Wandler angeschlossen werden. Beide Sensoren liefern Spannungswerte, die proportional zur gemessenen physikalischen Größe sind. Sie können als linear angenommen werden. Weitere Hinweise zu den Sensoren siehe unten (Datenblätter auf OLAT). Diese Sensoren sollen Sie regelmäßig auslesen und deren Werte umrechnen und anzeigen. Beachten Sie dabei die Datentypen der Werte (16 Bit), die ein Sensor liefert!

Die Feuchte soll in Prozent, die Temperatur (umschaltbar) in Grad Celsius oder Fahrenheit angezeigt werden. Die Temperaturwarnung soll die Überschreitung eines Schwellenwertes nach oben anzeigen. Eine erneute Warnung erfolgt erst wieder wenn: (1) die Warnung vom Benutzer gelöscht wurde und (2) die Temperatur unter die Schwelle gefallen und erneut darüber angestiegen ist.

Als Anzeige verwenden Sie ein 4x40 LCD-Display auf dem Sie die Tagesuhrzeit und Weckzeit im Format hh:mm:ss bzw. hh:mm anzeigen. Ferner die Temperatur (xxx), die Temperaturwarnschwelle (zzz) in der jeweils gewählten Einheit (C oder F) und die Feuchte (yy). Optimieren Sie die Displayansteuerung so, dass es möglichst „ruhig“ aussieht, also immer nur die notwendigen Ziffern ersetzt werden (kein komplettes Neuschreiben bei Wertänderung!). Die auffällige Gestaltung der beiden Alarmanzeigen ist Ihnen überlassen.

```
Uhrzeit     Weckzeit   Temperatur   Feuchte
hh:mm:ss     hh:mm     xxx°C (zzz)    yy%

   <Weckalarm>          <Temperaturalarm>
```

Zur Zeitmessung verwenden Sie einen der Timer, den Sie so konfigurieren, dass reale Zeit möglichst genau mittels Timer-ISR gezählt wird. Es genügt eine Sekunde als Zeitbasis (tick-Abstand) zu verwenden. Die Sensoren können Sie mit derselben Zeitbasis sekündlich oder seltener (z.B. alle 5 Sek auslesen und anzeigen. Wählen Sie die Zeit nicht zu lange, sonst dauert das Testen der Sensormessung lange).

Zur Einstellung der Uhrzeit, der Weckzeit und der Alarmtemperatur bekommen Sie einen Drehgeber. Für dessen Ablesung und Decodierung steht Ihnen ein Treiber (die Klasse QuadEncoder im Modul DigiPort) zur Verfügung. Hinweise dazu siehe unten. Wenn Sie Werte einstellen (hh, mm, ss, zzz), sollen diese sich unmittelbar auf dem LCD verändern (jeweils nur die notwendigen Ziffern!).

Die Alarme sollen auf dem LCD als Klartext und, wenn Sie möchten, zusätzlich auch per LED angezeigt werden. Blinken der LEDs und/oder des Textes wäre sicher sinnvoll.

Die Werte von Weckalarm und Temperaturalarm sind im EEPROM zu hinterlegen, sodass bei Neustart die zuletzt eingestellten Werte wieder verfügbar sind.

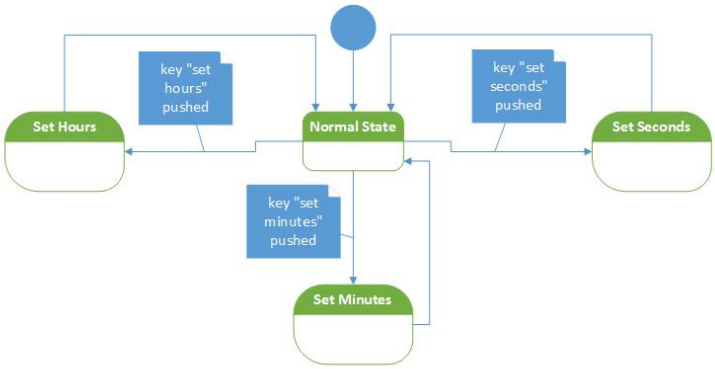
Programmieren Sie Tasten auf dem STK600 mit folgenden Funktionen:

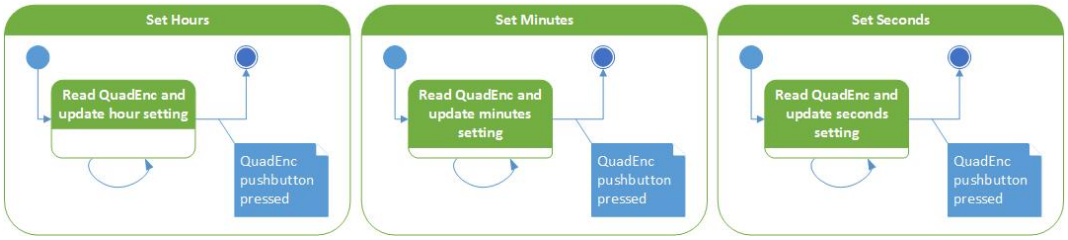
- „set hours“: Start der Einstellung der Stunden der Realzeituhr mittels Drehgeber
- „set minutes“: Start der Einstellung der Minuten der Realzeituhr mittels Drehgeber
- „set seconds“: Start der Einstellung der Sekunden der Realzeituhr mittels Drehgeber
- „set hours“: Start der Einstellung der Stunden der Weckzeit mittels Drehgeber
- „set minutes“: Start der Einstellung der Minuten der Weckzeit mittels Drehgeber
- „set temp threshold“: Start der Einstellung der Alarmtemperatur mittels Drehgeber
- Umschaltung zwischen °C und Fahrenheit-Anzeige (xxx und zzz)
- Bestätigung (Beenden) der Alarme

Der Drehgeber hat eine push-Funktion (eine Taste, die durch Drücken des Drehknopfs betätigt wird), die vom Treiber des Drehgebers gelesen werden kann. Durch Drücken soll jeweils der Einstellmodus verlassen und der momentane Wert übernommen werden. Da es beim Drücken dieses Tasters leicht zur Verstellung des Drehgebers kommt, sehen Sie als zusätzliche (besser zu bedienende) Alternative das Drücken irgendeiner Taste am STK600 vor.

Während Einstellungen vorgenommen werden muss die Uhr die timer-ticks weiter zählen – Sie dürfen aber ggf. die Aktualisierung des Displays vorübergehend suspendieren.

Das folgende Zustandsdiagramm zeigt bespielhaft, wie die Uhrzeit eingestellt wird (Weckzeit analog):





Offenbar benötigen Sie die „Read QuadEnc“ Funktion und die „set…“ Funktionen vielmals. Eine copy-paste Vervielfachung des Codes wird hier nicht akzeptiert – verwenden Sie möglichst wenige und möglichst kompakte Unterprogramme (mit entsprechenden Parametern)! Die Einstellung der Warntemperatur ist im Bild nicht gezeigt, würde aber in ähnlicher Weise aus „NormalState“ heraus ausgelöst werden.

Es ist Ihnen überlassen, ob Sie multi-tasking anwenden wollen! Jedenfalls: überlegen Sie sich eine einfache Programmstruktur! Die Verwendung von OO-Konzepten kann die Lösung besser zu strukturie-

ren helfen. Es ist bspw. eine gute Idee (das nur als Tipp nicht als Verpflichtung) die Uhr mit Weckfunktion und die Sensoren als Klassen zu definieren. Wie übersichtlich das dann im Hauptprogramm aussehen kann, zeigt das Listing am Ende dieses Dokuments (nur ein Denkanstoß).

Hinweise zum Quadratur-Encoder:

(Für weitergehende technische Erklärung siehe z.B.: http://www.mikrocontroller.net/articles/Drehgeber)

Für die Konfiguration und Ablesung des Quadratur-Encoders (QE) verwenden Sie ein Objekt der Klasse QuadEncoder. Dessen Konstruktor benötigt die Angabe des Ports, der zum Anschluss des QE verwendet werden soll – das ist im Labor immer Port **J**. Da die Klasse QuadEncoder Subklasse von DigiPortIRPT ist, generieren das Drehen des Knopfs oder das Drücken der Taste des Encoders Interrupts welche ohne Zeitverzug vom QuadEncoder Objekt gelesen und verarbeitet werden.

Zur Umrechnung der QE-Signale muss dem QuadEncoder-Objekt ein Wertebereich übergeben werden, den der Drehknopf durchlaufen soll. Dieser Bereich wird durch zwei int16_t Werte spezifiziert. Außerdem muss der Startwert gesetzt werden. Die Methode start() erledigt das in einem Schritt (Variante der Methode mit 3 Parametern). Wenn der Wertebereich schon gesetzt wurde und nicht verändert werden soll kann start() mit nur einem Parameter (dem Startwert) gerufen werden.

Der QE-Hardwarebaustein verfügt neben dem Drehknopf mit push-Funktion zusätzlich über drei LEDs in den Farben rot, grün und gelb. Wurde der Drehknopf lange genug nach links gedreht, wird das untere Ende des Wertebereichs erreicht. Dann leuchtet die gelbe LED. Wenn durch Rechtsdrehung das obere Ende erreicht wurde, leuchtet die rote LED. Sobald die push-Funktion ausgelöst wird leuchtet die grüne LED. Diese Anzeigefunktion erledigt das QuadEncoder-Objekt automatisch. Drehen über die Wertbereichsgrenzen hinaus bewirkt keine Werteänderung mehr.

Um die Werte des Drehgebers zu bekommen, fragt man das QuadEncoder-Objekt kontinuierlich ab und zeigt die Werte am Display an. Dabei kann man ermitteln, ob ein neuer Wert vorliegt (der noch nicht gelesen wurde); dazu dient die boole'sche Funktion new_value_available(). Wenn sie true liefert kann ein neuer Wert mit get_value() abgeholt werden. Das Objekt merkt sich, dass der Wert abgeholt wurde – new_value_available() würde nach get_value() solange false liefern bis der Drehknopf wieder gedreht wurde.

Mittels new_locked_value_available() kann festgestellt werden, ob die push-Funktion betätigt wurde und mit get_value() kann der zugehörige Wert gelesen werden.

Beispiel: Um den QE zur Uhrzeiteinstellung (Stunden) zu verwenden:

1. das QuadEncoder-Objekt mit dem zulässigen Wertebereich starten (0..23) und dabei
2. den Startwert setzen, indem Sie den aktuellen Stunden-Wert der Uhr übergeben
3. und dann solange den Wert lesen (sofern value_available true liefert) und am LCD anzeigen bis der Knopf zur Übernahme des Einstellwertes (oder eine Taste) gedrückt wird
4. jetzt verlassen Sie den Einstellmodus. Methode stop() des QE nicht vergessen; der QE wird ab sofort nicht mehr gelesen, die LEDs sind aus. Solange das System nicht in einem Einstellmodus ist, sollten Sie den QE auch nicht aktiviert halten oder auslesen.

Siehe auch die Doxygen Dokumentation zur Klasse QuadEncoder!

Hinweise zum AD Konverter (ADC) und den Sensoren:

Der verwendete Temperatursensor (ein LM335) ist ein analoges Bauteil, welches eine Ausgangsspannung proportional zu seiner Temperatur liefert. Man benötigt also einen Analog-Digital-Converter (ADC), der den analogen Spannungswert (in unserem Fall zwischen 0 und 5V = V_{REF}) in einen Zahlenwert wandelt. Der ATMega2560 hat 16 ADC eingebaut, die jeweils einen analogen Messwert in einen 10 Bit Digitalwert wandeln können (wir verwenden den „single-ended" Modus). Sehr vereinfacht gesagt approximiert der ADC den zu messenden Wert durch schrittweises Verändern einer Messspannung, bis diese dem zu messenden Wert annähernd entspricht. Die gezählten Schritte entsprechen dem gesuchten digitalen Wert. Als Treiber für den ADC steht Ihnen die Klasse ADConverter zur Verfügung. Pro zu messendem A/D Kanal instanziieren Sie ein Objekt dieser Klasse und binden es dabei an einen A/D-Kanal. Das Objekt können Sie dann bei Bedarf nach dem aktuellen Wert des Kanals fragen (er wird also auf Anfrage durch eine A/D Wandlung ermittelt).

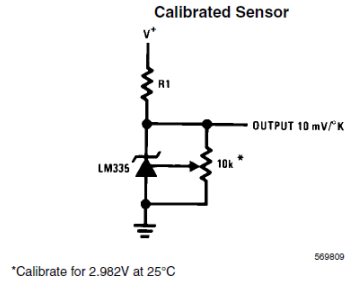
Temperatursensor und Feuchtesensor sind auf einer gemeinsamen Platine montiert und aufgrund ihrer Verdrahtung fest auf Kanal 3 (Temperatur) und Kanal 5 (Feuchte) bei Verwendung von **Port F** (im Labor ist hierfür immer dieser Port zu verwenden) angeschlossen. Wählen Sie beim Feuchtesensor als Averaging-Wert bspw. die 5, um entsprechend viele Messungen durch Mittelwertbildung zu einem „stabileren" Wert zu integrieren. Beim Temperatursensor brauchen Sie keinen Averaging-Wert angeben; es wird dann eine Messung ohne Mittelwertbildung verwendet.

Das Modul ADConverter zur Ansteuerung des ADC hat folgende wesentliche Funktionen:

```
// Instanziiere ein ADC Objekt für den Kanal chan. Die Kanäle 0 bis 7 gehören
// zum Port F und die Kanäle 8 bis 15 zum Port K. Mit avg kann für diesen Kanal
// die Anzahl der Wandlungen bestimmt werden, die per Mittelwertbildung
// zur Bildung eines geglätteten Messwertes herangezogen werden sollen.
// conv ist ein Funktionszeiger auf eine Funktion, die die Umrechnung der Werte
// des Sensors in konkrete phys. Größen erledigt; default: Identitätsfunktion.

ADConverter(uint8_t chan, uint8_t avg = 1, cft conv = identity);

// Hole den neuesten Wert vom ADC Objekt ab.

uint16_t get_value ();
```

Siehe auch die Doxygen Dokumentation zur Klasse ADConverter!

Zur Ermittlung der Umrechnungsformel des vom ADC gelieferten Wertes in einen Temperaturwert müssen Sie im Datenblatt des Temperatursensors sowie in der Beschreibung des ADC vom ATMega2560 (Abschnitt 26.7 „ADC Conversion Result", Seite 280) nachlesen. Die Kennlinie des Sensors (der wie im folgenden Bild gezeigt mit Kalibrierpoti beschaltet ist), hat eine Steigung von 10mV/°K, liefert ca. 2,98V bei 25°C und kann als linear angenommen werden. Implementieren Sie je eine Korrekturfunktion, die Sie statt der Identitätsfunktion in die ADConverter Objekte einhängen. Dann bekommen Sie automatisch korrigierte Werte von get_value() geliefert.



*Calibrate for 2.982V at 25°C

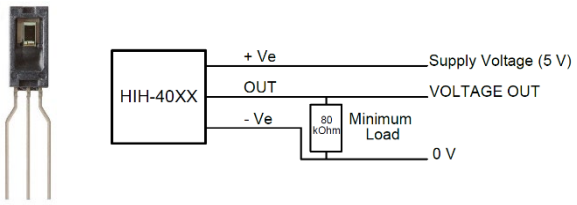
Temperature Accuracy (Note 1)

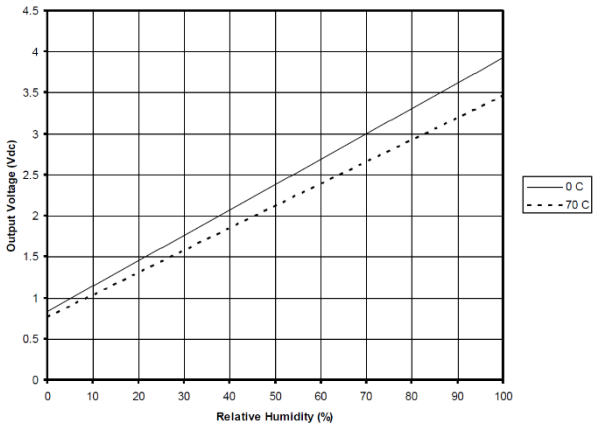
LM335, LM335A

Parameter	Conditions	LM335A			LM335			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage	T_C = 25°C, I_R = 1 mA	2.95	2.98	3.01	2.92	2.98	3.04	V
Uncalibrated Temperature Error	T_C = 25°C, I_R = 1 mA		1	3		2	6	°C
Uncalibrated Temperature Error	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		2	5		4	9	°C
Temperature Error with 25°C Calibration	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		0.5	1		1	2	°C
Calibrated Error at Extended Temperatures	$T_C = T_{MAX}$ (Intermittent)		2			2		°C
Non-Linearity	I_R = 1 mA		0.3	1.5		0.3	1.5	°C

Hinweise zum Feuchtesensor:

Der verwendete Feuchtesensor (ein HIH 4000) ist ein analoges Bauteil, welches eine Ausgangsspannung proportional zur Luftfeuchte liefert:





Generelle Hinweise zum Testat (notwendige Randbedingungen, die alle zu erfüllen sind):

- Es werden keine Gruppenabgaben akzeptiert.
- Dokumentieren Sie den Code durch maßvolle (sinnhafte) Kommentare.
- Bringen Sie außer dem lauffähigen Code einen Ausdruck zum Testat mit. Bei online Abgabe reicht Zusendung des Quellcodes per email.
- Dokumentieren Sie Struktur und Abläufe Ihres Programms geeignet (UML Diagramme, Flussdiagramme, Zustandsgraphen) und liefern Sie diese als PDF per email ab.
- Senden Sie mir Ihren Code und Ihre Dokumentation mindestens zwei Tage vor der Testatabnahme per email zu.
- Ich akzeptiere keine offensichtlich kopierten Lösungen! (Im Rahmen der Testatprüfung wird schnell ersichtlich, ob Ihre Lösung originär ist.)
- In der Testatprüfung werde ich abprüfen, ob Sie Ihre Lösung wirklich verstanden haben und Struktur und Funktionsweise im Detail erklären können. Je nach Corona-Situation erfolgt Befragung per ZOOM oder in Präsenz im Labor.

Ausschnitt aus der Musterlösung als „Denkanstoß“

```
…
//=============================================================================================
// define all  Objects
static DigiPortRaw  keys(PK,SET_IN_PORT);
static QuadEncoder  quenc(PJ);
static LCD          disp(PC, LCD_Type_40x4, WRAPPING_OFF|BLINK_OFF| CURSOR_OFF | DISPLAY_ON);
static Timer16      ticker(TC1, ticker_cb);
static myClock      theClock(&disp);
static mySensor     theSensors(&disp);

//=============================================================================================


int main(void)
{
    sei();
    ticker.start_ms(1000);
    disp.write_FLASH_text(headline);
    disp.set_pos(1,0);
    disp.write_FLASH_text(dataline);

    while (1)
    {
      if (do_update) {
          theClock.show();
          theSensors.show();
          do_update = false;
      }

      switch (keys.read_raw()) {
          case SET_H:     theClock.set_h(read_quenc(sethours, 1, 0, 2, 0, 23, theClock.get_h()));
                          break;
          case SET_M:     theClock.set_m(read_quenc(setminutes, 1, 3, 2, 0, 59, theClock.get_m()));
                          break;
          case SET_S:     theClock.set_s(read_quenc(setseconds, 1, 6, 2, 0, 59, theClock.get_s()));
                          break;
          case SET_AH:    theClock.set_ah(read_quenc(sethours, 1, 12, 2, 0, 23, theClock.get_ah()));
                          break;
          case SET_AM:    theClock.set_am(read_quenc(setminutes, 1, 15, 2, 0, 59, theClock.get_am()));
                          theClock.set_as(0);
                          break;
          case SET_TEMP:  theSensors.set_alarm_value(read_quenc(settempalarm, 1, 27, 3, 0, 50, theSensors.get_temp()));
                          break;
          case SET_UNIT:  theSensors.toggle_unit();
                          break;
          case CLR_ALARM: theClock.clear_alarm();
                          theSensors.clear_alarm();
                          break;
      }

    }
}

//=============================================================================================

static void ticker_cb(void){
    theClock.tick();
    do_update = true;
}

//=============================================================================================
// start quadencoder and show prompt on line 3 of display
// modify number at (line, pos) using len digits
// values range is min to max and initial value is start
// when done, clear prompt and return value upon key press
static uint8_t read_quenc (const char* prompt, uint8_t line, uint8_t pos, uint8_t len, uint8_t min, uint8_t max, int8_t start) { …
```