E-Paper Raumreservierung

Jannik Keßler

Ein Projekt im Rahmen des Moduls Sensoren und Aktoren.

Angewandte Informatik
Hochschule Fulda
Deutschland
07.09.2020

Inhaltsverzeichnis

1 Projektidee

Ziel des Projekts ist es mit Hilfe eines E-Papers eine Raumreservierung zu verwirklichen. Dabei geht es mir konkret um Meetingräume wie es sie in vielen Unternehmen aber bspw. auch in der Hochschulbibliothek gibt. Um vor Ort schnell zu erkennen ob der Raum momentan und vorallem in naher oder ferner Zukunft noch frei sein wird soll dieses E-Paper helfen. Das E-Paper soll mit Hilfe einer Smartphone App angesteuert werden um so den Raum reservieren zu können.
Dabei würde das E-Paper neben der Tür des Raumes hängen. Um sehen zu können wann der Raum frei ist soll auf dem Bildschirm eine Zeitleiste implementiert werden. Außerdem könnten die genauen Zeiträume eventuell noch ausgeschrieben angezeigt werden. Zusätzliche Erweiterungen sind denkbar.

2 Probleme & Lösungen

Probleme	Lösungen
Den Bildschirm ansteuern.	Die Bibliotheken verwenden die Sie mir geschickt haben [5, 9, 10].
Ich hatte Probleme mich in die Bibliotheken einzufinden.	Um das zu Lösen habe ich meine Fragen diesbezüglich gesammelt und an Sie geschrieben.
Erstellen von Apps.	Ihre Einführung in das Thema und die Vorstellung vom MIT App Inventor [6] haben diese Sache wesentlich vereinfacht.
Benutzen des partiellen Display Updates.	Aus zeitlichen Gründen leider nicht mehr mit ins Projekt aufgenommen.
Stringbefehle zum Testen des Displays per seriellem Monitor verwenden.	Es hat etwas Zeit gekostet die korrekte Implementierung zu erreichen, sodass die Funktionen des String Objekts, Compiler und seriellem Monitor sich nicht mehr in die Haare bekommen haben.
Von der App die Daten/Stundenzahlen an das E-Paper schicken	Ich sende nun per String die Daten zum Display, wobei ich aber darauf achten muss, dass dabei von der Software an den String noch ein Whitespace Zeichen angehängt wird.
Das Logo als Bitmap auf das E-Paper zu bekommen.	“Lvgl” hat Probleme gemacht, also habe ich einen Bitmap Converter ohne “Lvgl” gesucht und die Bitmap einfach so verwendet.

3 Durchführung

3.1 Schritt für Schritt

1. Zunächst habe ich mit den Examples aus der Bibliothek [10] und dem von Ihnen geschickten Sketch experimentiert. Daraus entstand ein erstes eigenes "Hello World".

2. Ich lese mich in Deep Sleep ein [2].

3. Zu Testzwecken verwende ich den seriellen Monitor um per Befehl [8] das Display zu testen.

4. Graphisches Feedback um zu erkennen wann der Raum reserviert ist auf das Display implementieren.

5. Anschließend Prototyp BLE Funktionalität für das Display entwickeln und testen per nRF Connect App. Dabei habe ich mich an BLE Sketches von ihrer Website orientiert [7].

6. Eine eigene App entwickeln welche den gewünschten Zeitraum per Stunden an das E-Paper schicken soll. Bei dieser App habe ich mich ebenfalls von einer App von Ihnen orientiert [7].

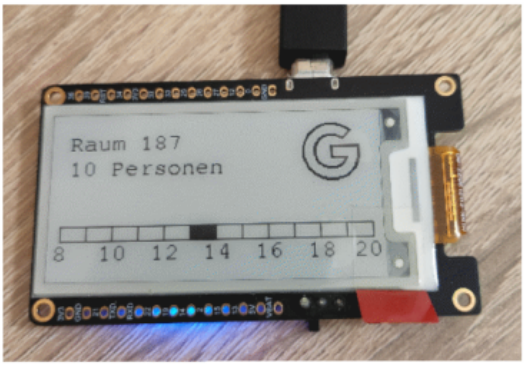
7. Vom Display ein Feedback an die App schicken, ob der Raum frei oder bereits reserviert ist implementiert.

8. Weitere Fehlerbehandlungen hinzugefügt, überwiegend auf Seiten der App. Leider musste ich nach einer längeren Arbeitspause feststellen, dass die BLE Funktionalität bei neu installierter App komischerweise sehr lange braucht um das E-Paper zu finden und sich damit zu verbinden. Vor der Pause (Juli 2020) ging das noch sehr schnell.

9. GUI verschönern indem die Zeitleiste übersichtlicher gestaltet wird. Außerdem habe ich eine Raumnummer, Personenanzahl und Logo [4] der Firma hinzugefügt [3]. Anstelle die genauen Zeiträume ausgeschrieben auf das Display zu bannen, habe ich mich dazu entschieden den Raum genauer zu beschreiben und ein Firmenlogo hinzuzufügen.

10. Deep Sleep über einen 5 Sekunden Timer eingebaut [1]. Dabei wird das Display nur dann neugeladen wenn es auch eine neue Verbindung gibt.

3.2 Aufbau



Links sieht man die App mit dem Verbindungsauf- und abbau Button. Darunter wird eine Statusmeldung über den Zustand der Verbindung angezeigt. In dem Textfeld kann man seine gewünschte Zeit eingeben und per weiterem Button abschicken. Darunter befindet sich eine weitere Statusmeldung darüber ob die Reservierung funktioniert hat. **Rechts** ist das Display nach der App Eingabe abgebildet. Wie man sehen kann ist der Block von 13 bis 14 Uhr schwarz markiert, was bedeutet, dass der Raum zu diesem Zeitraum belegt ist. Auch ist die Raumnummer, Anzahl Sitzplätze und ein obligatorisches Firmenlogo implementiert.

3.3 Codebeispiele

3.3.1 setup()

Hier wird das E-Paper erstmal initialisiert. Zuerst wird dafür der BLE Service erstellt und advertised. Dann wird beim ersten Start oder falls gerade eine neue Verbindung aufgebaut wurde das Display initialisiert/aktualisiert. Und wenn es keine Verbindung zu einem Gerät gibt, geht das E-Paper in den DeepSleep. Das E-Paper soll ohne Verbindung 5 Sekunden in den DeepSleep gehen. Ich arbeite ohne die Loop aber mit Callbacks.

```
void setup() {
  Serial.begin(115200);
  Serial.println("# starte ble server");
  // ble initialisieren
  BLEDevice::init("ESP-SRV");
  // ble server erzeugen
  BLEServer *pServer = BLEDevice::createServer();
  // server callbacks installieren
  pServer->setCallbacks(new MyServerCallbacks());
  // service erzeugen
  BLEService *pService = pServer->createService(SERV_UUID);
  // characteristic erzeugen und einstellen welche Properties es haben soll.
  pCharacteristic = pService->createCharacteristic(
        CHAR_UUID,
        BLECharacteristic::PROPERTY_READ |
        BLECharacteristic::PROPERTY_WRITE |
        BLECharacteristic::PROPERTY_NOTIFY |
        BLECharacteristic::PROPERTY_INDICATE);
  // descriptor erzeugen - fuer notify/indicate notwendig. Fuer die "Pipe"
    zustaendig schaetze ich.
  pCharacteristic->addDescriptor(new BLE2902());
  // service starten
  pService->start();
  pCharacteristic->setCallbacks(new bleCharacteristicCallback());

  // advertising starten
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERV_UUID);
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x06);
  pAdvertising->setMinPreferred(0x12);
  BLEDevice::startAdvertising();
  Serial.println("# ble server gestartet. warte auf anrufe ...");
  delay(5000);    //Ansonsten wird die naechste Methode uebersprungen.
  if (firstStart || newConnection) {
    initDisplay();
    firstStart = false;
    newConnection = false;
  }
  Serial.println("Enter Command: ");

  if (!isConnected){
    startSleep();
  }
}
```

3.3.2 bleCharacteristicCallback

Sobald eine Nachricht von der App bzw. dem Smartphone an das E-Paper gelangt wird diese Nachricht ausgelesen und weiterverarbeitet. Dabei ist zu beachten, dass die Nachrichtenlänge um 1 größer

ist, da der MIT AppInventor wohl an den zu versendenten String etwas dranhängt. Nach dem erkannt wurde um welche gewünschte Zeit es sich handelt wird diese an die Funktion bookroom() weitergegeben. Es gibt auch die Möglichkeit die Reservierungen des Raums komplett zu löschen indem man “free” eingibt.

```
class bleCharacteristicCallback: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    std::string msg = pCharacteristic->getValue();
    Serial.printf("msg: %s|\n", msg.c_str());
    //Serial.flush();       //Um Buffer zu leeren.
    delay(5000);
    //App Input Auswertung
    Serial.printf("Laenge: %d\n", msg.length());
    if (msg.at(0) == 'f' && msg.at(1) == 'r' && msg.at(2) == 'e' && msg.at(3) == 'e
    '/*msg.compare("free") == 0*/) {   //String so pruefen, weil die App ein mir
    unbekanntes Whitespace anhaengt.
      Serial.println("FREE");
      freeRooms();
    }
    //Z.B. "8 9" ; Laenge von AI2 um 1 groesser.
    else if (msg.length() == 4) {
      Serial.printf("%d | %d\n", int(msg.at(0)), int(msg.at(2)));
      bookRoom(int(msg.at(0)) - 48, int(msg.at(2)) - 48);     //-48 because of char
     to int
    }
    //z.B. "9 16"
    else if (msg.length() == 5) {
      char buf[3] = {msg.at(2), msg.at(3)};
      Serial.printf("%d | %d\n", int(msg.at(0)) - 48, atoi(buf));
      bookRoom(int(msg.at(0)) - 48, atoi(buf));
    }
    //z.B. "19 20"
    else if (msg.length() == 6) {
      char buf[3] = {msg.at(0), msg.at(1)};
      char buf1[3] = {msg.at(3), msg.at(4)};
      Serial.printf("%d | %d\n", atoi(buf), atoi(buf1));
      bookRoom(atoi(buf), atoi(buf1));
    }
  }
};
```

3.3.3 bookRoom()

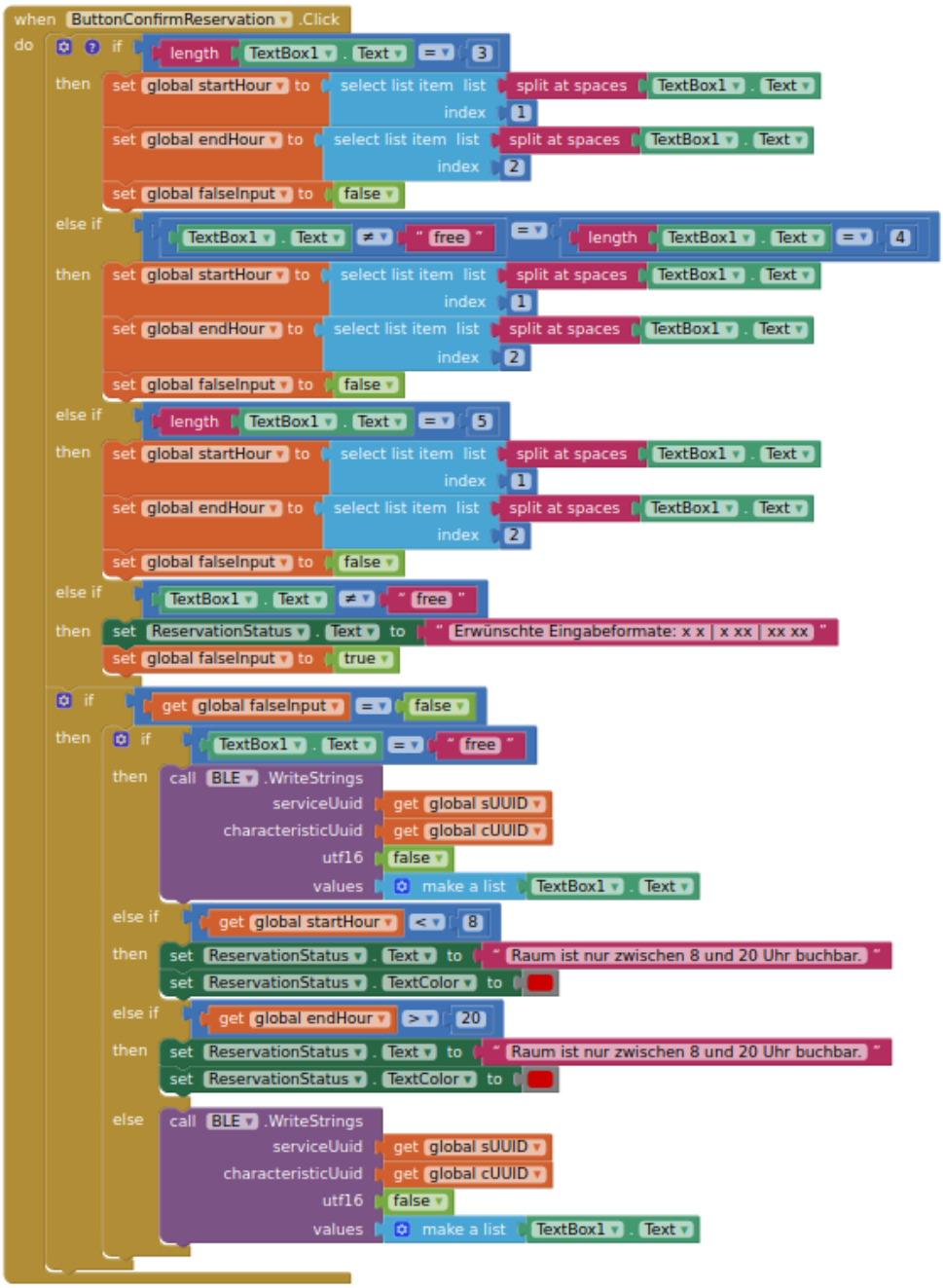
Es wird erstmal überprüft ob die Eingabe im korrekten Zeitrahmen liegt. Wenn das der Fall ist, wird ein Array period erstellt, welches die zu reservierenden Blöcke halten soll. Dabei werden die zu reservierenden Blöcke von 0 bis 12 benannt und in period eingefügt(im Falle einer Reservierung von 8 bis 20 Uhr). Dann wird dieses Array mit dem im Speicher hinterlegten Array roomBooked verglichen, welches die bereits reservierten Blöcke hält. Zum Schluss wird der App noch ein Feedback per Notify zugesendet und das Display aktualisiert.

```
//Von 8 Uhr morgens bis 20 Uhr abends
void bookRoom(uint8_t startHour, uint8_t endHour) {
  // Matches restrictions
  Serial.printf("Zu buchender Zeitraum: %d | %d\n", startHour, endHour);
  if ((startHour >= 8 && startHour <= 20) && (endHour >= 8 && endHour <= 20)) {
        uint8_t  period[endHour - startHour];
        uint8_t periodSize = sizeof(period) / sizeof(period[0]);
        //writes booking periods in form of roomBooked(0 - 12); damit es leichter
    zu vergleichen ist
```

```
        for (int i = 0; i < periodSize; i++) {
          period[i] = (startHour + i) - 8;
        }
        //Is room still free?
        for (int i = 0; i < periodSize; i++) {
          //If true then not free
          if (roomBooked[period[i]]) {
            if (isConnected) {
              pCharacteristic->setValue("1");
              pCharacteristic->notify();
            }
            Serial.println("Belegt!");
            return;
          }
        }
        //App Feedback fuer erfolgreiche Buchung
        pCharacteristic->setValue("0");
        pCharacteristic->notify();
        Serial.println("set+notify");
        delay(1000);
        //Book the room
        for (int i = 0; i < periodSize; i++) {
          roomBooked[period[i]] = true;
        }
        initDisplay();
  }
  else {
    Serial.printf("Reservierung nur von 8-20 Uhr moeglich");
  }
}
```

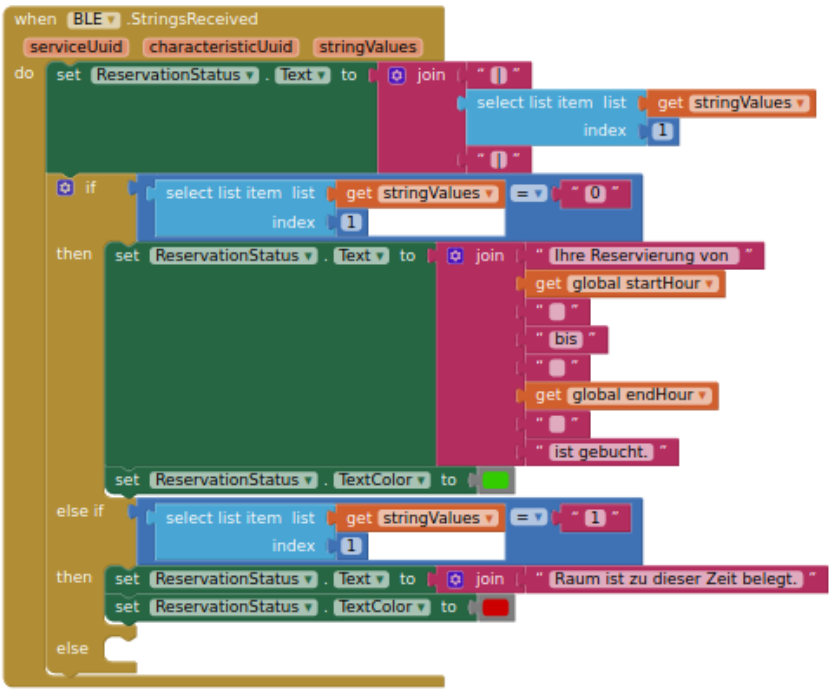
3.3.4 App: Nachricht abschicken

Sobald man auf den “Reservierung abschicken”-Button klickt wird gecheckt ob es sich um “free” oder um eine tatsächliche Reservierung handelt. Im Falle einer korrekten Eingabe wird ein String aus der eingegebenen Nachricht erstellt und an das E-Paper geschickt.



3.3.5 App: Feedback verarbeiten

Wenn vom E-Paper der Code “0” zurückkommt dann war die Reservierung erfolgreich. Und wenn der Code “1” ist, ist der Raum zu der Zeit schon vergeben was dann ebenfalls in der App angemerkt wird.



4 Feedback zum Projekt

Ich habe das Projekt als sehr angenehm empfunden, vorallem weil man sich individuell ausleben konnte. Das hat die Langzeitmotivation gestärkt. Ich konnte viel über die Funktionsweise von BLE sowie die von E-Papers lernen. Auch der MIT AppInventor war mir neu, weshalb ich nun eine schnelle Möglichkeit kenne Apps zu prototypen. Alles in allem war das Projekt erfolgreich, auch wenn ich gerne mehr Zeit investiert hätte, was aber aufgrund von anderen Modulen nicht ganz so einfach ist.

Literatur

[1] DeepSleep. https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/.

[2] Espressif Seite über Deep Sleep. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html.

[3] Github Bitmap-Editor. https://javl.github.io/image2cpp/.

[4] Google Logo. https://icons8.de/icon/17904/google-logo.

[5] Herstellerseite des E-Papers. http://www.lilygo.cn/prod_view.aspx?TypeId=50031&Id=1149&FId=t3:50031:3.

[6] MIT App Inventor. https://appinventor.mit.edu/.

[7] Modul Website. http://c.rz.hs-fulda.de/.

[8] Serieller Monitor Befehle. https://www.norwegiancreations.com/2017/12/arduino-tutorial-serial-inputs/.

[9] lewisxhe. TTGO-EPaper-Series. https://github.com/lewisxhe/TTGO-EPaper-Series.

[10] ZinggJM. Gxepd2. https://github.com/ZinggJM/GxEPD2.