Technische Informatik IV: Praktikum

Protokoll zu Aufgabe A9

von Max Wisniewski, Alexander Steen

Vorbereitung

Zur Vorbereitung diente der Punkt *Hardware* auf der Veranstaltungsseite, der *ADL User Guide* (Abschnitte *AT Commands* und *SMS Service*), sowie das *AT Command Interface* (Abschnitt 10 *Time Management Commands*, Abschnitt 7 *Embedded module Status Commands*).

Aufgaben

1. Tastenbenachrichtigungen verarbeiten
2. Benachrichtigungen über Tastendrücke mit Uhrzeit versehen
3. Benachrichtigungen über Tastendrücke per SMS verschicken

Dokumentation

AT+CMER Dieses Kommando ermöglicht es spontane Nachrichten über getätigte Tastendrücke zu (de-)aktivieren. Die Syntax ist:
`AT+CMER=[<mode>][,[<keyp>][ ,...]]`
Dabei ist `mode` ein Parameter, der die interne Nachrichtenverarbeitung beeinflusst, was für unsere Zwecke allerdings nicht weiter von Wichtigkeit ist. `keyp` entscheidet, ob spontane Nachrichten bei Tastendruck geschickt werden. Ein Wert von 0 deaktiviert die Nachrichten, eine 1 aktiviert sie. Es gibt noch weitere Parameter, auf die wir hier nicht eingehen.
Betätigt man bei eingeschalteter Benachrichtigung eine Taste, erhält man eine spontane Nachricht der Form:
`+CKEV: <Taster>,<Art>`
Wobei `Taster` die Nummer der Taste ist. Der Parameter `Art` beschreibt, ob die Taste gedrückt oder losgelassen wurde und hat dann den Wert 1 bzw. 0.

AT+CCLK Mit Hilfe dieses Kommandos kann die aktuelle Zeit herausgefunden bzw. gesetzt werden. Die Zeit setzt man mit `AT+CCLK=<Datums-Zeit-Zeichenkette>`, die aktuelle Zeit findet man mit `AT+CCLK?` heraus. Auf letzteres erhält man als Antwort eine spontane Nachricht der Art
`+CCLK: <Datums-Zeit-Zeichenkette>`

adl_atCmdCreate Diese Funktion erlaubt es, ein AT Kommando programmatisch an das Modul zu schicken und die Antwort durch eine Callback-Funktion bearbeiten zu lassen. Die Syntax ist:
`s8 adl_atCmdCreate ( ascii * kommando, u16 rspflag, adl_atRspHandler_t handler, ... );`
Dabei bezeichnet `kommando` den AT-Kommando-String, der ausgeführt werden soll. Mit `rspflag` kann man genauert steuern, an welchen internen Port das kommando weitergeleitet werden soll. Ein Wert von `FALSE` verhindert, dass Antworten an andere Applikationen weitergeleitet werden. `handler` bezeichnet die Callback-Funktion, die sich dann um die Verarbeitung der Antworten kümmert. Am Ende des Aufrufes werden nach alle Strings aufgelistet, auf die der angegebene Handler lauscht. Diese Liste muss mit `NULL` beendet werden. Die Callback-Funktion muss folgendem Prototyp entsprechen:
`bool(*) adl_atRspHandler_t (adl_atResponse_t *Params)`[1]

[1] Das struct adl_atResponse_t wird nicht genauer beschrieben, da dies die Dokumentaton sprengen würde. Man findet eine ausführliche Beschreibung auf Seite 65 des ADL User Guide.

adl_atUnSoSubscribe Registriert eine Handler-Callbackfunktion auf eine bestimme spontane Nachricht. Die Syntax ist:
`s16 adl_atUnSoSubscribe (ascii * UnSostr, adl_atUnSoHandler_t UnSohdl );`
Dabei ist `UnSostr` der Name der spontanen Nachricht und `UnSohdl` die Callbackfunktion, die folgendem Prototyp entsprechen muss:
`bool (* adl_atUnSoHandler_t) (adl_atUnsolicited_t *)`[2]

wm_strGetParameterString Liefert eine Zeichenkette eines bestimmten Parameters aus einer AT-Antwort.
`ascii * wm_strGetParameterString (ascii * dst, const ascii * src, u16 Position );`
Dabei ist `dst` der Zielpuffer, in dem der Parameter gespeichert wird, `src` die Quelle (AT-Antwort) und `Position` die Angabe, welcher von evtl. mehreren Parametern zurückgeliefert werden soll.

Durchführung und Auswertung

Es gibt zwei verschiedene Antworten die das Modul ausgibt:

Response sind direkte Antworten auf eine Interaktion mit dem Modul, z.B. Ausführung eines AT-Kommandos. Diese Nachrichten werden sofort ungepuffert ausgegeben.

Unsolicited Message sind Antworten, denen keine Aufforderung hervorgehen musste und dementsprechend unvorhersehbar aufkommen können. Diese Nachrichten werden ausgegeben, wenn spezielle Aktionen (Broadcast-Nachricht, Tastendruck,...) ausgelöst worden sind bzw. erkannt wurden. Diese Nachrichten können gepuffert werden und können daher evtl. mit Verzögerung angezeigt werden.

```
AT+CMER=,1
OK
```

Wir schalten die Benachrichtigungen für Tastenbetätigungen via Konsole an.

Testlauf:

```
+CKEV: 0,1
+CKEV: 0,0
+CKEV: 1,1
+CKEV: 1,0
+CKEV: 2,1
+CKEV: 2,0
+CKEV: 3,1
+CKEV: 3,0
```

Hier haben wir jede Taste einmal ausprobiert. Jeweils die erste Ziffer bezeichnet die Tastennummer. Wir bekommen pro Taste zwei Meldungen, da wir sowohl für das Drücken als auch für das Loslassen eine Benachrichtigung erhalten.

```
AT+CCLK="10/04/27,13:04:31"
OK
```

Wir stellen die Uhrzeit des Moduls auf die aktuelle Zeit.

```
bool keyhandler(adl_atUnsolicited_t *paras)     {
    adl_atSendResponse(ADL_AT_RSP, "\r\nEs wurde eine Taste gedrueckt\r\n");
    return false;
}
```

[2]Das struct adl_atUnsolicited_t wird nicht genauer beschrieben, da dies die Dokumentaton sprengen würde. Man findet eine ausführliche Beschreibung auf Seite 49 des ADL User Guide.

Dieser Handler gibt einen String aus, wenn er aufgerufen wird. Damit dies bei einem Tastendruck passiert, registrieren wir ihn im Folgenden.

```
void main_task(void) {
    // Tasten-Nachrichten einschalten. Kein Handler benötigt.
    adl_atCmdCreate("AT+CMER=,1", FALSE, NULL, NULL);
    // Handler auf die spontane Nachricht des Tastendrucks registrieren.
    adl_atUnSoSubscribe("+CKEV: ", (adl_atUnSoHandler_t) keyhandler);
}
```

Nun wird bei jedem Tastendruck der obige Handler ausgeführt.

Testlauf:

```
+GSM: Anmeldung im Netz abgeschlossen
Es wurde eine Taste gedrueckt
Es wurde eine Taste gedrueckt
```

Hier haben wir eine Taste ausprobiert. Wir bekommen zwei Meldungen, da der Handler sowohl für das Drücken als auch für das Loslassen angesprochen wird.

Nun erweitern wir die Funktionalität um eine Benachrichtigung auf Taster 3 mit Uhrzeitangabe. Dafür legen wir uns zunächst eine globale Flag an, die anzeigt, ob die Taste gerade gedrückt bzw. losgelassen wurde.

```
bool pressed = FALSE;

bool datehandler(adl_atResponse_t *paras)   {
    // Ausgabepuffer für die komplette Ausgabe
    ascii puffer[50];
    // Puffer für den Datums-String
    ascii datepuffer[20];
    // Hole Datum aus Handler-Parameter
    wm_strGetParameterString(datepuffer, paras->StrData, 1);
    // Baue String zusammen
    wm_sprintf(puffer,
        ((pressed) ? ("Kontakt geschlossen am %s \r\n")
                   : ("Kontakt geöffnet am %s \r\n")), datepuffer);

    adl_atSendResponse(ADL_AT_RSP, puffer);
    return FALSE;
}

bool keyhandler(adl_atUnsolicited_t *paras) {
    ascii param1[1];
    // Parameter auslesen (Taste gedrückt oder losgelassen?)
    wm_strGetParameterString(param1, paras->StrData, 2);
    // ... global speichern
    pressed = wm_atoi(param1);
    // Zeit anfordern und datehandler für die Antwort bereitstellen
    adl_atCmdCreate("AT+CCLK?", FALSE,  datehandler, "+CCLK:", NULL);
    return FALSE;
}
```

Wir schreiben nun also einen Handler (`keyhandler`), der ausschließlich bei einem Druck von Taste 3 (siehe unten) ausgesprochen wird. Dieser findet nur heraus, ob die Taste gedrückt oder losgelassen wurde und deligiert dann an den Handler für die Uhrzeit (`datehandler`) weiter. In datehandler wird dann der endgültige Ausgabestring zusammengebaut.

```
void main_task(void) {
    // Tasten-Nachrichten einschalten. Kein Handler benötigt.
    adl_atCmdCreate("AT+CMER=,1", FALSE, NULL, NULL);
    // Keyhandler speziell auf Taste 3 reagieren lassen.
    adl_atUnSoSubscribe("+CKEV: 3", (adl_atUnSoHandler_t) keyhandler);
}
```

Nun reagiert der Handler nur auf die Taste drei.

Testlauf:

```
+GSM: Anmeldung im Netz abgeschlossen
Kontakt geschlossen am 11/05/11,10:52:23
Kontakt geöffnet am 11/05/11,10:52:25
Kontakt geschlossen am 11/05/11,10:52:27
Kontakt geöffnet am 11/05/11,10:52:28
```

Hier haben wir die Funktionalität zwei Mal ausprobiert.

Als letzten Schritt wird nun das Programm so erweitert, dass die Meldung (nur die Kontakt-Geschlossen-Meldung) via SMS verschickt wird. Dafür speichern wir nun zusätzlich die von `adl_smsSubscribe` zurückgelieferte `smsHandle` global, da diese von `adl_smsSend` benötigt wird.

```
bool pressed = FALSE;
s8 smsHandle;
```

An `keyhandler` ändert sich nichts. In datehandler wird nun kein String mehr ausgegeben, sondern eine SMS geschickt.

```
bool datehandler(adl_atResponse_t *paras) {
    // Ausgabepuffer für die komplette Ausgabe
    ascii puffer[50];
    // Puffer für den Datums-String
    ascii datepuffer[20];
    // Hole Datum aus Handler-Parameter
    wm_strGetParameterString(datepuffer, paras->StrData, 1);
    // Baue String zusammen
    wm_sprintf(puffer,
        ((pressed) ? ("Kontakt geschlossen am %s \r\n")
                   : ("Kontakt geöffnet am %s \r\n")), datepuffer);

    // statt ausgeben, SMS senden - aber nur beim Schließen des Kontaktes
    if (pressed) {
        if (OK != adl_smsSend(smsHandle, "0171XXXXXXX",
                              puffer, ADL_SMS_MODE_TEXT)) {
            // Falls nicht erfolgreich: Fehler ausgeben
            adl_atSendResponse(ADL_AT_RSP, "Fehler beim Senden.\r\n");
        }
    }
    return FALSE;
}
```

Damit das funktioniert, müssen wir noch zusätzlich SMS-Handler registrieren. Diese müssen nichts tun, aber dennoch vorhanden sein, also definieren wir sie entsprechend der Prototypen:

```
bool leersmshandler(ascii * tel, ascii * smstime, ascii * text) {
    return true;
}
void leersmsCtrlHandler(u8 event, u16 nb) {}
```

Nun können wir mit einer kleinen Erweiterung der anfänglichen Registrationen die Funktionalität bereitstellen:

```
void main_task(void) {
    // SMS-Handler registrieren
    smsHandle = adl_smsSubscribe(leersmshandler,
                        leersmsCtrlHandler, ADL_SMS_MODE_TEXT);
    // Tasten-Nachrichten einschalten. Kein Handler benötigt.
    adl_atCmdCreate("AT+CMER=,1", FALSE, NULL, NULL);
    // Keyhandler speziell auf Taste 3 reagieren lassen.
    adl_atUnSoSubscribe("+CKEV: 3", (adl_atUnSoHandler_t) keyhandler);
}
```

Einen Testlauf können wir hier offensichtlich nicht bereitstellen - aber wir versichern: Es funktioniert. Ganz ehrlich. Wirklich ;-)