Technische Informatik IV: Praktikum

Protokoll zu Aufgabe A10

von Alexander Steen, Max Wisniewski

Vorbereitung

Zur Vorbereiung haben wir uns unter Hardware angesehen, welche Bezeichner die I/O Ports für die LEDs haben (obwohl dies durch das Framework von led.h komplett unnötig geworden ist). Wir haben uns wie empfohlen im Kapitel 20 im *ADL Commands Interface Guide* über die Befehle *CLIR* und *CLIP*, sowie im *ADL User Guide* Kapitel 3.10 über den GPIO Service (obwohl dies wieder durch led.h überflüssig war) informiert.

Aufgaben

- Entwickeln Sie eine Anwendung mit folgenden Eigenschaften
  - Alle Anrufer werden abgewiesen. Jeder Anruf wird mit **ath** beendet.
  - Nur wenn der Anruf von einer fest eingespeicherten Rufnummer erfolgt, wird die LED2 eingeschaltet und die LED3 soll für die Dauer von 20 Sekunden im Sekundentakt blinken.
  - Nach Ablauf der 20 Sekunden soll ein **Voice Call** an die fest eingespeicherte Nummer erfolgen. Die LED2 wird mit dem Ende des Anrufs ausgeschaltet.
  - Erweitern Sie die Funktionalität in der Form, dass ein im Telefonbuch vorhandener Eintrag "Alarm" genauso behandelt wird, wie die fest eingespeicherte Nummer.
    In einem zweiten Schritt erlauben Sie einer SMS, die ein Passwort und eine Nummer erhält, den Eintrag unter "Alarm" auf die neue Nummer zu ändern. Das Passwort muss mit einem fest abgespeicherten Passwort übereinstimmen.

Dokumentation

AT+CLIP Mit diesem Befehl sorgt man dafür, dass die Rufnummer des Anrufers angezeigt wird (und gegebenenfalls mit einem Telefonbucheintrag assoziiert wird). Diesen Befehl haben wir bereits im Protokoll zu Aufgabe A3 beschrieben. Für eine genauere Erläuterung kann man an dieser Stelle nachlesen.

AT+CLIR Mit diesem Befehl, kann man dafür sorgen, ob die eigene Rufnummer unterdrückt wird. Mit dem Befehl *AT+CLIR=<n>* kann man den Modus einstellen, nachdem man operieren möchte. Dabei kann n die Werte 0 (default Wert, benutzt einen Indikator der vorm CLIR Service, der momentan vom Netz bereitgestellt wird, verwendet), 1 (schaltet CLIR ein, die Rufnummer wird unterdrückt) oder 2 (CLIR wird unterdrückt, die Rufnummer wird mitgeschickt) annehmen.
Den Wertebereich, kann man mit *AT+CLIR=?* abfragen und erhält die Antwort *+CLIR=(0-2)*.
Um die aktuelle Einstellung abzulesen, kann man *AT+CLIR?* verwenden. Als Antwort erhält man *+CLIR: <n>,<m>* wobei n eigenen Modus beschreibt (sie aktion von CLIR) und m den Modus anzeigt, den das Netz bereitstellet.
m kann dabei die Werte 0 (CLIR wird nicht unterstützt), 1 (CLIR wird nicht permanent unterstützt [Mode 2 disabled]), 2 (Netzwerk unbekannt), 3 (Modus 0 ist eingeschränkt) und 4 (Modus 0 ist uneingeschränkt erlaubt)

GPIO Service Der GPIO Service ist, wie die meisten der Funktionen über verschiedene Methoden und Callbackmethoden anzusprechen. Wir müssten in diesem Fall die Methode *adl_ioWrite* verwenden und gegebenenfalls *adl_ioSubscribe*, aber der Umgagng mit den Feinheiten, die durch eine Vielzahl von structs und handlern gegeben ist, wird uns durch die Klasse *led.h* abgenommen. Dort sind schon

Implementierungen zum Ansprechen der LEDs gegeben. Man kann dort die LEDs einfach über den Befehl *led_on(u8 led)* und *led_off(u8 led)* ansprechen. Dabei muss man nicht einmal die Bezeichener für die LEDs (GPIO19-GPIO22) angeben, sondern nur noch eine Nummer, wie sie in der Beschreibung gegeben wurde. (LED2 = 2, LED3 = 3)

Call Service Den Call Service haben wir schon in Protokoll zu Aufgabe A6 / A7 beschrieben. Wir verweisen an dieser Stelle auf das alte Protokoll.

Timer Den Timer haben wir schon in Protokoll zu Aufgabe A6 beschrieben. Wir verweisen an dieser Stelle auf das alte Protokoll.

led_init Mit dieser Funktion sorgt man dafür, dass die LEDs ansprechbar werden. Weitere Voraussetzung ist, dass man die LED-Flag in der *config.h* gesetzt hat.

led_on Dies ist eine einfache Funktion, die das ganze Ansprechen der I/O Ports für die LEDs abnimmt. Auch die genaue Bezeichnung der LEDs ist unnötig, da diese in der led.h einfach von Integer auf die Namen gemappt werden. (Bezeichnungen, wie weiter unten Beschrieben immer LED0 : 0, LED1 : 1, usw.)

led_off Analog zur ***led_on*** Funktion, schaltet diese Funktion die LED wieder aus. Parameter ist genau so ein Integer, wie bei **led_on**.

Durchführung

Als erstes haben wir, um die LEDs über den eingebundenen Headerfile ansprechen zu können, in der config.h den Wert von LED auf 1 gesetzt, damit die I/O-Ports für das Modul beim Start aktiviert werden. Der erste

Part der Aufgabe, war recht leicht umzusetzen. Wir haben ersteinmal einen *CallHandler* eingerichtet, der ganz simpel den Anruf beendet.

```
1  s8 callHandler(u16 event, s32 callId)
2      {
3      if (event == ADL_CALL_EVENT_RING_VOICE)
4          {
5          adl_callHangup();
6          }
7      ...
8      return OK;
9      }
10
11 void main_task(void) {
12     ...
13     adl_callSubscribe((adl_callHdlr_f) callHandler);
14     }
```

Wir richten den callHandler ein und fragen in diesem ab, ob wir von einem **Voice Call** angerufen wurden. Ist dies der Fall, legen wir, wie in der Aufgabe verlangt einfach auf. Der nächste Aufgabenteil sah vor, dass

wir, wenn wir von einer fest eingespeicherten Nummer angerufen werden, die LED2 anschalten und die LED3 20 Sekunden lang jede Sekunde einmal blinken lassen.

```
adl_tmr_t * ledTimeoutHandle;
u8 timeout = 0;
bool ledOn = FALSE;
ascii * telNum = "+49152--------";

void ledTimeoutTimer(u8 id, void * context)
    {
    if (timeout-- > 0)
        {
        ledOn ? led_off(3) : led_on(3);
        ledOn = !ledOn;
        adl_atSendResponse(ADL_AT_RSP,"3 : an\r\n");
        }
    else
        {
        adl_tmrUnSubscribe(ledTimeoutHandle, ledTimeoutTimer, ADL_TMR_TYPE_100MS);
        ledOn = FALSE;
        led_off(3);
        adl_atSendResponse(ADL_AT_RSP,"3 : aus\r\n");
        }
    }

bool clipHandler(adl_atUnsolicited_t * data)
    {
    ascii nummerpuffer[20];
    wm_strGetParameterString(nummerpuffer, data->StrData, 0);

    if (wm_strcmp(nummerpuffer, telNum) == 0)
        {
        adl_atSendResponse(ADL_AT_RSP, "max hat angerufen!\r\n");
        led_on(2);
        adl_atSendResponse(ADL_AR_RSP,"2 : an\r\n");
        // neuen timer mit timeout 20 starten
        timeout = 20;
        ledTimeoutHandle = adl_tmrSubscribe(TRUE, 10, ADL_TMR_TYPE_100MS, ledTimeoutTimer);
        }
    return FALSE;
    }

void main_task(void)
    {
    led_init();
    adl_atCmdSend("AT+CLIP=1", NULL, NULL);
    adl_atUnSoSubscribe("+CLIP:", clipHandler);
    adl_callSubscribe((adl_callHdlr_f) callHandler);
    }
```

Wir rufen hier als erstes die led_init() auf, damit wir die LEDs ansprechen können. Als nächstes senden wir den Befehl ab, dass die CLIP Funktion auf 1 gesetzt werden soll, d.h. wir wollen wissen, wer uns anruft. Da uns die Antwort nicht interessiert, geben wir keine Handler für die Antwort rein.

Als nächstes richten wir den Handler für die allgemeine Nachricht von *+CLIP* ein. Diese Nachricht wird bei jedem Anruf einmal abgefeuert und enthält die Nummer vom Anrufenden.

Der letzte Befehl ist derselbe wie eben und richtet unseren callHandler ein. In der clipHandler Funktion nehmen wir uns als erstes den 1. Parameter der Antwort und überprüfen, ob es der eingespeicherten Nummer (global *telNum*) entspricht. Sind die beiden Nummern gleich schicken wir eine Debugmeldung ab, das wir angerufen wurden, schalten die LED2 ein und starten einen neuen zyklischen Timer, der jede Sekunde einmal unseren ledTimeouHandler aufruft. Der ledTimeoutHandler dekrementiert nun den globalen Zähler timeout und vergleicht, ob dieser größer als 0 ist. Solange er größer ist, schaltet er (mittels einer globalen Variable ledOn) den LED3 in jedem Schritt an, bzw. aus.

Ist der timeout auf 0 gefallen, schalten wir die LED3 aus. An dieser Stelle schließt sich die nächste Aufgabe an.

```
void ledTimeoutTimer(u8 id, void * context)
    {
    if (timeout-- > 0){...}
    else
        {
        adl_tmrUnSubscribe(ledTimeoutHandle, ledTimeoutTimer, ADL_TMR_TYPE_100MS);
        ledOn = FALSE;
        adl_atSendResponse(ADL_AT_RSP,"3 : aus\r\n");
        led_off(3);
        adl_callSetup(telNum, ADL_CALL_MODE_VOICE);
        }
    }

s8 callHandler(u16 event, s32 callId)
    {
    if (event == ADL_CALL_EVENT_RING_VOICE){...}
    else if (event == ADL_CALL_EVENT_NO_CARRIER)
    {
        led_off(2);
        adl_atSendResponse(ADL_AT_RSP,"2 : aus\r\n");
    }
    return OK;
    }
```

Um zu erreichen, dass wir nach den 20 Sekunden anrufen, müssen wir nur 2 unserer schon geschriebenen Fukntionen leicht modifizieren.

Als erstes schreiben wir in den *callHandler*, dass wir bei einer Nachricht *ADL_CALL_EVENT_NO_CARRIER*, die wir standardmäßig erhalten, wenn unser Gesprächspartner auflegt, die LED2 ausschalten.

Der Anruf ist auch kein Thema. Wir haben eben schon die Stelle gesehen, an der wir die LED3 ausschalten. An dieser Stelle setzten wir nur noch einen Anruf an die eingespeicherte Nummer ab. Damit haben wir schon die Funktionalität eines Rückrufes nach 20 Sekunden erledigt. Im nächsten Part mussten wir das ganze noch so erweitern, dass wir auch eine Telefonnummer aktzeptieren, die in unserem Telefonbuch unter "Alarm" steht.

Unser Ansatz ist etwas simpel gestaltet. Im Groben sahen wir zwei Möglichkeiten der Herangehensweise: In der einen testen wir jedes Mal bei einem Anruf, wie der momentane Eintrag unter "Alarm" aussieht. Dies führte in der Überlegung zu einigen Koordinationsproblemen, mit dem Timer. Da man die LEDs nur an (oder an/aus) schalten soll, wenn der Anruf von der richtgen Nummer kommt. Allerdings würde sich durch die Abfrage auf das Telefonbuch der ganze vorgang verzögern. Der zweite Ansatz, den wir gewählt haben, sah vor, die Rufnummer beim Start des Programms einmal auszulesen, in einer Variable zu speichern. Jedesmal, wenn die Nummer sich ändert, wird diese Variable aktualisiert. Dies kann der Fall sein, wenn jemand den Eintrag von außen ändern, dazu brauchten wir einen Handler für *+CPBW* auf den aktuellen Platz unserer Nummer. Und der einfachere Fall, wenn wir über die SMS die Nummer ändern.

```
ascii alarmNum[20];
bool isAlarm = FALSE;
ascii * passwort = "geheim";

bool smsHandler(ascii * tel, ascii * time, ascii * text)
{
    ascii commandBuffer[50];
    if (wm_strlen(text) < 6) return true;
    if (wm_strncmp(passwort, text, 6) == 0){
        // passwort korrekt
        wm_strncpy(alarmNum, text+7, 20);
        wm_sprintf(commandBuffer, "AT+CPBW=42,\"%s\",145, \"Alarm\"",alarmNum);
        adl_atCmdSend(commandBuffer, NULL, NULL);
        return false;
    } else {
        // normale sms oder pw inkorrekt, weiterleiten
        return true;
    }
}
void smsCtrlHandler(u8 event, u16 mb) {return;}

```

```
22 bool alarmHandler(adl_atResponse_t * params)
23     {
24     wm_strGetParameterString(alarmNum, params->StrData, 2);
25     adl_atSendResponse(ADL_AT_RSP, alarmNum);
26     return false;
27     }
28
29 void ledTimeoutTimer(u8 id, void * context)
30     {
31     if (timeout-- > 0){...}
32     else
33         {
34         ...
35         adl_callSetup(isAlarm ? alarmNum : telNum, ADL_CALL_MODE_VOICE);
36         }
37     }
38
39 bool clipHandler(adl_atUnsolicited_t * data)
40     {
41     ascii nummerpuffer[20];
42     wm_strGetParameterString(nummerpuffer, data->StrData, 0);
43     isAlarm = wm_strcmp(nummerpuffer, alarmNum) == 0;
44
45     if (wm_strcmp(nummerpuffer, telNum) == 0 ||  isAlarm)
46         {...}
47     return FALSE;
48     }
49
50 void main_task(void)
51     {
52     adl_atCmdSend("AT+CPBF=\"Alarm\"", alarmHandler, "+CPBF:",NULL);
53     adl_smsSubscribe(smsHandler, smsCtrlHandler, ADL_SMS_MODE_TEXT);
54     ...
55     }
```

Wir haben 3 neue globale Variablen angelegt. In der ersten speichern wir die Nummer, die unter "Alarm" steht. In der zweiten speichern wir uns, ob wir von "Alarm" oder der fest gespeicherten Nummer angerufen wurden und in der letzten steht das Passwort, das man zum ändern der "Alarm" Nummer braucht. Der reine Umbau auf die alternative Nummer ist nicht schwierig. Wir fragen zu Beginn in der main ab, wie die Nummer lautet. Im *alarmHandler* reagieren wir einfach auf die Antwort und speichern die Nummer in der globalen Variable.

Nun gucken wir bei der *+CLIP* Nachricht einfach nach, ob die anrufende Nummer mit "Alarm" oder der festen Nummer übereinstimmt. Ist sie alarm, wird dies gleich in der globalen Vairable gespeichert.

Am Ende von *timerHandler* können wir nun einfach testet, ob *isAlarm* gesetzt ist. Danach Rufen wir entweder die feste oder die "Alarm" Nummer an. Im *smsHandler* testen wir, ob die ersten 6 Zeichen (unser Passwort ist 6 Zeichen lang) dem Passwort entsprechen. Ist dies der Fall kopieren wir ab Stelle 7 (wir erwarten ein Trennzeichen zwischen Passwort und neuer Nummer) und kopieren die nächsten 20 Zeichen in den Speicher und aktualisieren den Eintrag im Telefonbuch. Mit dieser geringfügigen Veränderung können wir nun auch auf "Alarm" reagieren.

Auswertung

Aufgrund der wenig textuellen Ausgabe, werden wir diesmal viel von unserem Versuch berichten. Und anzeigen, was dabei zu sehen war.

```
RING
+CLIP: "0151543-----",161
```

In diesem Versuch haben wir von einer nicht gespeicherten Nummer angerufen. Wie man sieht, bekommt man den RING und die Nummer wird angezeigt. Als nächstes zeigen wir die Ausgabe, wenn wir von einer der eingespeicherten Nummern aus anrufen.

```
RING
2 : an
3 : an
3 : aus
3 : an
...
3 :an
3 : aus

2 : aus
```

Wir sehen hier an erster Stelle das *RING*, aber die Information über die Nummer nicht mehr. Gleich darauf springen unsere Debugnachrichten an, die uns im Sekundentakt die LED3 ein und ausschalten (die LED selber ging auch an und aus), und nach 20 Sekunden blieb sie aus. Mit einer kurzen Verzögerung haben wir einen Anruf bekommen. Die zweite LED geht zu Beginn an und ging, nachdem wir aufgelegt haben aus. Es bleibt zu erwähnen, dass wenn man den Anruf nicht nimmt, das die LED2 erst mit einigen Minuten Verzögerung ausgeht. Wir haben nicht heraus bekommen warum, aber scheinbar bekommt das Modul die Nachricht erst mit einiger Verzögerung.

Als nächstes haben wir eine SMS an das Modul geschickt. Wir haben die Nachricht darüber wieder abgefangen, aber in der zweiten Abfrage der eingespeicherten Nummern, sieht man, dass sich der Eintrag unter *ALARM* auf eine neue Nummer (die, die wir mitgeschickt haben) geändert hat.

```
1  at+cpbr=1,100
2  +CPBR: 2,"017665316700",129,"HWP2"
3  +CPBR: 3,"017665338235",129,"HWP3"
4  +CPBR: 4,"017665348409",129,"HWP4"
5  +CPBR: 5,"017683269836",129,"Wojtek"
6  +CPBR: 6,"017665390997",129,"HWP6"
7  +CPBR: 7,"017668065207",129,"HWP7"
8  +CPBR: 8,"017665374344",129,"HWP8"
9  +CPBR: 10,"+4915118985237",145,"dasandere"
10 +CPBR: 11,"017665376367",129,"dieda"
11 +CPBR: 42,"+49176-------",145,"Alarm"
12 OK
13
14 //An dieser Stelle haben wir eine SMS geschrieben
15 //mit dem richtigen Passwort
16
17 at+cpbr=1,100
18 +CPBR: 2,"017665316700",129,"HWP2"
19 +CPBR: 3,"017665338235",129,"HWP3"
20 +CPBR: 4,"017665348409",129,"HWP4"
21 +CPBR: 5,"017683269836",129,"Wojtek"
22 +CPBR: 6,"017665390997",129,"HWP6"
23 +CPBR: 7,"017668065207",129,"HWP7"
24 +CPBR: 8,"017665374344",129,"HWP8"
25 +CPBR: 10,"+4915118985237",145,"dasandere"
26 +CPBR: 11,"017665376367",129,"dieda"
27 +CPBR: 42,"0151534------",145,"Alarm"
28 OK
```