Technische Informatik IV: Praktikum

Protokoll zu Aufgabe A6

von Max Wisniewski, Alexander Steen

Vorbereitung

Zur Vorbereitung der Programmierung haben wir uns mit der *C-Einführung OpenAT* sowie dem *ADL User Guide* auseinandergesetzt. Für diese Aufgabe ist dabei insbesondere die Sektion *Timers* von Relevanz.

Aufgaben

1. Programm schreiben, welches einen Text über die serielle Schittstelle ausgibt.
2. Programm modifizieren, sodass es den Text zehnmal ausgibt.
3. Programm modifizieren, sodass es den Text zehnmal mit einem Abstand von jeweils 10 Sekunden ausgibt

Dokumentation

adl_atSendResponse Diese Funktion sendet eine Nachricht über die serielle Schnittstelle.
`s32 adl_atSendResponse(u16 Typ, ascii * Nachricht)`
wobei der Nachrichten-Typ für unsere Zwecke entweder `ADL_AT_UNS` für spontane Nachrichten oder `ADL_AT_RSP` für Antworten ist. Damit wird der Text der Nachricht an die Standardausgabe geschickt.

adl_tmrSubscribe Diese Funktion registriert einen Timer. Folgende Parameter sind zu verwenden:
`adl_tmr_t * adl_tmrSubscribe(bool Zyklisch, u32 Verzögerung, adl_tmrType_e Timer-Typ, adl_tmrHandler_t Handler)`
Wird der Timer Zyklisch angelegt (Parameter `true`) so registriert sich der Timer nach einem Aufruf der Callback-Funktion mit derselben Verzögerung erneut. Bei einem Timer-Typ von `ADL_TMR_TYPE_100MS` ist die Verzögerung in 100 ms anzugeben.
Als Rückgabe erhält man einen Verweis auf den angelegten Timer. Der übergebene Handler muss vom Typ
`void (*) adl_tmrHandler_t (u8 Id, void * Context)`
sein, wobei die `Id` ein Verweis auf den Timer ist, der gerade abgelaufen ist.

adl_tmrUnSubscribe Diees Funktion erlaubt es einem, einen Timer wieder zu deregistrieren.
`adl_tmrUnSubscribe(adl_tmr_t * Timer, adl_tmrHandler_t Handler, adl_tmrType_e Timer-Typ)`
Dabei ist Timer der von der Subscribe-Funktion zurückgegebene Verweis auf den Timer.

Durchführung und Auswertung

Wir wählen für die folgenden Nachrichten den Nachrichten-Typ `ADL_AT_RSP` aus, da dieser auf jeden Fall schnell auf dem Endgerät angezeigt wird.

```
void main_task (void) {
    adl_atSendResponse (ADL_AT_RSP, "\r\nHollari-dudoedel-di\r\n");
}
```

Testlauf

```
...
+GSM: Anmeldung im Netz abgeschlossen
Hollari-dudoedel-di
```

Im Folgenden realisieren wir die mehrfache Ausgabe derselben Nachricht über eine while-Schleife.

```
void main_task(void) {
    u16 zaehler = 0;
    while (zaehler++ < 10) {
        adl_atSendResponse(ADL_AT_RSP, "\r\nHollari-dudoedel-di\r\n");
    }
}
```

Testlauf

```
...
+GSM: Anmeldung im Netz abgeschlossen
Hollari-dudoedel-di

Hollari-dudoedel-di

Hollari-dudoedel-di

...
```

Um den Timer nach der richtigen Anzahl von Ausgaben zu entfernen, erstellen wir uns eine globale Variable `zaehler`, die die bisher ausgeführten Ausgaben zählt. Außerdem speichern wir global den Verweis auf den Timer. Diesen Verweis benutzen wir in dem Handler `ausgabe`, um uns nach der zehnten Ausgabe abzumelden.

```
u16 zaehler = 0;
adl_tmr_t *timer_n;

void ausgabe(u8 event, void *context) {
    if (zaehler++ < 10) {
        adl_atSendResponse(ADL_AT_RSP, "\r\nHollari-dudoedel-di\r\n");
    } else {
        adl_tmrUnSubscribe(timer_n, ausgabe, ADL_TMR_TYPE_100MS);
    }
}
```

Wir melden den Timer an:

```
void main_task(void) {
    timer_n = adl_tmrSubscribe(
        TRUE, // zyklisch
        100, ADL_TMR_TYPE_100MS, // 100 * 100ms = 10s
        ausgabe);
}
```

Testlauf

```
...
+GSM: Anmeldung im Netz abgeschlossen
Hollari-dudoedel-di

Hollari-dudoedel-di

Hollari-dudoedel-di

...
```

Um einen Timer anzumelden, der einmalig nach 20 Sekunden seinen Handler ausführen soll, benutzen wir

```
adl_tmrSubscribe(FALSE, 200, ADL_TMR_TYPE_100MS, Handler)
```