

# Protokoll 2

## Praktikum Technische Informatik

Tas Soti, Richard Wilhelm, Naja v. Schmude

22. Mai 2008

### 1 Die erste eigene Anwendung mittels OpenAT

#### 1.1 Vorbereitung

Als erstes muss sich mit den Timern unter OpenAT vertraut gemacht werden. Dabei gibt es zwei wichtige Funktionen, einmal `adl_tmrSubscribe` und `adl_tmrUnSubscribe` zum anderen.

Mit der ersten kann man einen Timer erschaffen und mit der letzteren einen noch laufenden Timer abmelden. Die genauere Funktionalität wird bei den folgenden Aufgaben deutlich.

#### 1.2 Aufgaben und Durchführung

- *Es soll über die serielle Schnittstelle ein beliebiger Text einmalig ausgegeben werden.*

---

```
/* Diese Methode gibt einen Text einmalig über die
   serielle Schnittstelle aus. */
void einfacheAusgabe() {
    adl_atSendResponse(ADL_AT_RSP, "\r\nDas ist unser
                             anständiger Text.\r\n\r\n");
}
```

---

Mit dem Funktionsaufruf `adl_atSendResponse` wird die Nachricht, die als zweiter Parameter übergeben wird, über die serielle Schnittstelle verschickt und auf der Konsole sichtbar.

Der erste Parameter gibt die Art der Nachricht ein, durch `ADL_AT_RSP` wird z.B. sofort die Nachricht ausgegeben. Weitere Parameter wären hier z.B. noch `ADL_AT_UNSUB` für eine spontane Nachricht oder `ADL_AT_INT`.

- *Der Text soll 10 mal hintereinander im Abstand von 1 Sekunde ausgegeben werden.*

---

```
/* Ein Timer wird erstellt, der bis auf weiteres jede
   Sekunde die Funktion timer_Handler aufruft. */
void zehnfacheAusgabe() {
    timer = (adl_tmr_t *) adl_tmrSubscribe(TRUE,
        timer_period, ADL_TMR_TYPE_100MS, (adl_tmrHandler_t)
        timer_Handler);
}

/* 10mal wird die Methode aufgerufen und der Text
   ausgegeben, dann wird der Timer abgemeldet */
void timer_Handler (u8 Id) {
    if(a6counter<10) {
        adl_atSendResponse(ADL_AT_RSP, "\r\nDas ist unser 10
        mal-Test.\r\n");
        a6counter++;
    } else {
        adl_tmrUnSubscribe(timer, timer_Handler,
            ADL_TMR_TYPE_100MS);
    }
}
```

---

Durch den Aufruf von `zehnfacheAusgabe()` in der `Apply`-Methode wird zunächst der Timer `timer` erstellt mittels `adl_tmr_Subscribe`. Dabei gibt das `TRUE` des ersten Parameters an, dass es sich um einen zyklischen Timer handelt, d.h. dass sich nach jedem Ablauf des Timers automatisch ein neuer startet, bis man ihn abmeldet.

Mit dem zweiten und dritten Parameter wird definiert, wie lange der Timer laufen soll. Und zwar stehen zwei verschiedene Timer-Typen zu Verfügung: Es gibt den 100ms- (`ADL_TMR_TYPE_100MS`) und den 18,5ms-Timer (`ADL_TMR_TYPE_TICK`). Um nun auf das rechte Zeitmaß zu kommen, muss also noch entsprechend des Timer-Typs mit einem Faktor multipliziert werden. Der Faktor wird dabei als zweiter Parameter angegeben, der Timer-Typ wird mit dem dritten definiert. In unserem Fall ist die Variable `timer_period` auf 10 gesetzt, so dass wir mit dem 100ms-Timer auf eine Sekunde kommen.

Nach Ablauf der Sekunde wird dann die Methode aufgerufen, die im vierten Parameter spezifiziert ist, hier die Methode `timer_Handler`.

In dieser Methode wird dann die eigentliche Ausgabe stattfinden. Hierbei ist die Variable `a6counter` die Zählvariable, die sich die Anzahl der Durchläufe merkt, und solange der Wert noch nicht 10 ist munter den Text "Das ist uner 10mal-Test." ausgibt.

Beim 11. Durchlauf wird dann der Timer abgemeldet. Der erste Parameter gibt hier den Timer an, welcher gestoppt werden soll, dann die Funktion, die immer aufgerufen wurde und nochmals der Typ des Timers.

- Mit welchen Parametern muss man einen 100ms-Timer erzeugen, wenn dieser in seinem Handler einmalig einen Text nach 20 Sekunden ausgeben soll?

Nach obiger Erläuterung der Parameter ist dies schnell zu beantworten. Es darf sich um

keinen zyklischen Timer handeln, und um auf 20 Sekunden zu kommen braucht man den Faktor 200 für einen 100ms-Timer. Demnach lautet der Aufruf für einen Handler `timer_Handler`

---

```
adl_tmrSubscribe(FALSE, 200, ADL_TMR_TYPE_100MS, (  
    adl_tmrHandler_t) timer_Handler);
```

---

Wenn man nun die Funktion `a06_apply()` aufruft, die erst die `einfacheAusgabe()` und dann die `zehnfacheAusgabe()` aufruft, wird folgende Ausgabe im Terminal erscheinen.

```
// Modul-Initialisierung: Gerät funktionsbereit!
```

```
Das ist unser anständiger Text.
```

```
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.  
Das ist unser 10mal-Test.
```

## 2 Das erste eigene AT-Kommando

### 2.1 Vorbereitung

Als erstes muss sich natürlich in den entsprechenden Stoff eingelesen werden. Zum Erstellen eines eigenen Kommandos ist der Befehl `adl_atCmdSubscribe` zuständig. Er erwartet als Parameter den Namen des Kommandos und wie schon bei den Timern gesehen den Methodennamen einer Funktion, die dann angibt, was der Befehl eigentlich leisten soll. Zusätzlich wird noch durch die Konstanten `ADL_CMD_TYPE_TEST`, `ADL_CMD_TYPE_READ`, `ADL_CMD_TYPE_ACT`, `ADL_CMD_TYPE_PARA` definiert, ob auf den Befehl mit 1. "=?", 2. "?" 3. ohne jegliche Parameter, 4. beliebiger Anzahl von Parametern reagiert werden soll. Alle die Parameter repräsentieren übrigens eine 16bit Zahl, die per OR Verknüpfung beliebig kombiniert werden können. Die minimale und maximale Anzahl der Parameter wird dann zusätzlich noch als 16bit Zahl rangeknüpft.

Nachdem nun die Funktionsweise der Parameterdefinition verstanden wurde, kann mit den Aufgaben begonnen werden.

## 2.2 Aufgaben und Durchführung

- Es soll ein AT-Kommando erstellt werden, das bei AT+COUNTDOWN, ein OK zurückgibt.

---

```
adl_atCmdSubscribe("AT+COUNTDOWN", countdown,
    ADL_CMD_TYPE_ACT);
----
/* Funktion, die aufgerufen wird, wenn AT+COUNTDOWN benutzt
   wird. Sie bestimmt das Verhalten dieses Kommandos */
void countdown() {
    adl_atSendResponse(ADL_AT_RSP, "\r\nOK\r\n");
}
```

---

Wie bereits beschrieben wird also in der main-Methode das Kommando definiert, welches "AT+COUNTDOWN" heißen soll. Es ruft beim Eingang von dem Befehl (da nur ADL\_CMD\_TYPE\_ACT angegeben wird, wird nur auf den reinen Befehl reagiert) dann die Methode countdown() auf, die dann einfach im Terminal ein OK ausgibt.

- Erweitern Sie das Kommando, so dass es genau einen Parameter akzeptiert, der die Anzahl der Sekunden zum Runterzählen angibt. Wie oben erwähnt soll der Zählerstand im Sekundentakt ausgegeben werden. Nach Ablauf soll es eine besondere Meldung geben.

---

```
adl_atCmdSubscribe("AT+COUNTDOWN", countdown,
    ADL_CMD_TYPE_ACT | ADL_CMD_TYPE_PARAM | 0x0011);
----
/* Funktion, die aufgerufen wird, wenn AT+COUNTDOWN benutzt
   wird. Sie bestimmt das Verhalten dieses Kommandos */
void countdown(adl_atCmdPreParser_t *parameter) {
    if (parameter->Type == ADL_CMD_TYPE_ACT) { // AT+
        COUNTDOWN
        adl_atSendResponse(ADL_AT_RSP, "\r\nOK\r\n");
    }
    else if (parameter->Type == ADL_CMD_TYPE_PARAM) { // AT+
        COUNTDOWN=<parameter>
        param1[64];
        wm_strGetParameterString(param1, parameter->StrData,
            1);
        adl_atSendResponse(ADL_AT_RSP, "\r\nOK\r\n");
        a7counter = wm_atoi(param1);
        timer = (adl_tmr_t *) adl_tmrSubscribe(TRUE, 10,
            ADL_TMR_TYPE_100MS, (adl_tmrHandler_t)
            countdown_Timer_Handler);
    }
}
```

---

```

/* Timer-Handler, der von dem AT+COUNTDOWN Befehl zum
   herunterzählen benutzt wird. */
void countdown_Timer_Handler (u8 Id) {
    ascii out[64];
    if(a7counter > 0) { // Counter ist noch nicht bei null
        angekommen
        wm_sprintf(out, "+COUNTDOWN_□%d\r\n", a7counter);
        adl_atSendResponse(ADL_AT_RSP, out);
        a7counter--;
    } else if (a7counter <= 0) { // Counter ist bei null
        angekommen, stoppe ihn
        adl_tmrUnSubscribe(timer, countdown_Timer_Handler,
            ADL_TMR_TYPE_100MS);
        adl_atSendResponse(ADL_AT_RSP, "\r\nFertig!\r\n");
    }
}

```

---

Als erstes muss der Befehl zum Registrieren des Kommandos entsprechend erweitert werden, mit Hinzufügen von ADL\_CMD\_TYPE\_PARA werden Parameter akzeptiert und das 0x0011 besagt, dass mindestens ein Parameter da sein muss und maximal auch, also genau einer. Wir müssen jetzt entsprechend die countdown-Methode ändern. Als erstes führt man eine Abfrage ein, um was für einen Typ von Befehl (also mit oder ohne Parameter) es sich beim aktuellen Aufruf handelt mittels Zugriff auf parameter->Type, wobei parameter bei Angabe im Methodenkopf automatisch mit übergeben wird. Im Parameterfall wird nun durch den Aufruf von wm\_strGetParameterString(param1, parameter->StrData, 1) der übergebene Parameter an Stelle 1 in das Feld param1 übertragen. Dieser Wert liegt ja nur als String vor, muss also noch als Integer geparkt werden und wird dann in der globalen a7counter Variablen gespeichert. Da ein Countdown realisiert werden soll, muss jetzt ein entsprechender Timer registriert werden, der dann solange a7counter größer als null ist die entsprechende Nachricht ausgibt und den Counter runterzählt, bis er null ist. Dann wird der Timer wieder abgemeldet und Fertig! ausgegeben.

- Die Eingabe von AT+COUNTDOWN? soll den ursprünglich eingestellten Zahlenwert ausgeben. Sollte gerade kein Countdown aktiv sein, dann sollte eine entsprechende Meldung erscheinen.

---

```

adl_atCmdSubscribe("AT+COUNTDOWN", countdown,
    ADL_CMD_TYPE_READ | ADL_CMD_TYPE_ACT |
    ADL_CMD_TYPE_PARA | 0x0011);
----
/* Funktion, die aufgerufen wird, wenn AT+COUNTDOWN benutzt
   wird. Sie bestimmt das Verhalten dieses Kommandos */
void countdown(adl_atCmdPreParser_t *parameter) {
    ...
}

```

```

else if (parameter->Type == ADL_CMD_TYPE_PARA) { // AT+
    COUNTDOWN=<parameter>
    ...
    a7counter = wm_atoi(param1);
    a7counterStartValue = a7counter;
    timer = (adl_tmr_t *) adl_tmrSubscribe(TRUE, 10,
}
else if (parameter->Type == ADL_CMD_TYPE_READ) { // AT+
    COUNTDOWN?
    ascii out[64];
    if(a7counterStartValue == 0)
        adl_atSendResponse(ADL_AT_RSP, "\r\nThere is no
        countdown...\r\n");
    else {
        wm_sprintf(out,"%d\r\n",a7counterStartValue);
        adl_atSendResponse(ADL_AT_RSP, out);
    }
}
}

```

Jetzt wird noch ADL\_CMD\_TYPE\_READ hinzugefügt, damit das mit dem Fragezeichen klappt. Zusätzlich muss natürlich die countdown-Methode entsprechend erweitert werden. Zum einen ist eine globale Variable a7counterStartValue hinzugekommen, die beim Registrieren eines Countdowns sich den Anfangswert merkt. Dieser kann dann in dem Zweig, der für den ADL\_CMD\_TYPE\_READ zuständig ist, ausgelesen werden (falls er ungleich null ist, dies ist nämlich die Startbelegung) und ausgegeben. Andernfalls bekommt man eine entsprechende Info.

- *Es soll sich bei aktivem Countdown kein neuer starten lassen.*

---

```

/* Timer-Handler, der von dem AT+COUNTDOWN Befehl zum
   herunterzählen benutzt wird. */
void countdown_Timer_Handler (u8 Id) {
    ascii out[64];
    if(a7counter > 0) { // Counter ist noch nicht bei null
        angekommen
        wm_sprintf(out,"+COUNTDOWN_%d\r\n", a7counter);
        adl_atSendResponse(ADL_AT_RSP, out);
        countingActive = TRUE;
        a7counter--;
    } else if (countingActive && a7counter <= 0) { //
        Counter ist bei null angekommen, stoppe ihn
        adl_tmrUnSubscribe(timer, countdown_Timer_Handler,
            ADL_TMR_TYPE_100MS);
        countingActive = FALSE;
        a7counterStartValue = 0;
        adl_atSendResponse(ADL_AT_RSP, "\r\nFertig!\r\n");
    }
}

```

```

    }
}

/* Funktion, die aufgerufen wird, wenn AT+COUNTDOWN benutzt
   wird. Sie bestimmt das Verhalten dieses Kommandos */
void countdown(adl_atCmdPreParser_t *parameter) {
    if (parameter->Type == ADL_CMD_TYPE_ACT) { // AT+
        COUNTDOWN
        adl_atSendResponse(ADL_AT_RSP, "\r\nOK\r\n");
    }
    else if (parameter->Type == ADL_CMD_TYPE_PARA) { // AT+
        COUNTDOWN=<parameter>
        if (countingActive) { // ein Countdown läuft bereits
            adl_atSendResponse(ADL_AT_RSP, "\r\nCountdown_
                _cannot_start_a_new_one!\r\n");
        }
        else { // kein Countdown aktiv, kann einer also
            gestartet werden
            param1[64];
            wm_strGetParameterString(param1, parameter->StrData,
                1);
            adl_atSendResponse(ADL_AT_RSP, "\r\nOK\r\n");
            a7counter = wm_atoi(param1);
            a7counterStartValue = a7counter;
            timer = (adl_tmr_t *) adl_tmrSubscribe(TRUE, 10,
                ADL_TMR_TYPE_100MS, (adl_tmrHandler_t)
                countdown_Timer_Handler);
        }
    }
    else if (parameter->Type == ADL_CMD_TYPE_READ) { // AT+
        COUNTDOWN?
        ascii_out[64];
        if (a7counterStartValue == 0)
            adl_atSendResponse(ADL_AT_RSP, "\r\nThere_
                _is_no_
                _countdown_...\r\n");
        else {
            wm_sprintf(out, "%d\r\n", a7counterStartValue);
            adl_atSendResponse(ADL_AT_RSP, out);
        }
    }
}
}

```

---

Das ist jetzt die endgültige Version. Es wurde noch eine globale countingActive Variable eingeführt, die immer dann gesetzt wird, wenn eine Countdown gestartet wird. So wird verhindert, dass sich mehrere Starten lassen, in dem an entsprechenden Stellen diese abgeprüft wird.

Die Ausgabe eines Testlaufs sieht dann wie folgt aus:

```
// Modul-Initialisierung: Gerät funktionsbereit!  
AT+COUNTDOWN  
OK
```

```
AT+COUNTDOWN=10  
OK  
+COUNTDOWN 10  
+COUNTDOWN 9  
+COUNTDOWN 8  
+COUNTDOWN 7  
AT+COUNTDOWN?  
10
```

```
+COUNTDOWN 6  
+COUNTDOWN 5  
+COUNTDOWN 4  
AT+COUNTDOWN=5  
Countdown active, cannot start a new one!
```

```
+COUNTDOWN 3  
+COUNTDOWN 2  
+COUNTDOWN 1  
Fertig!
```

```
AT+COUNTDOWN=5  
OK  
+COUNTDOWN 5  
+COUNTDOWN 4  
+COUNTDOWN 3  
+COUNTDOWN 2  
+COUNTDOWN 1  
Fertig!
```