

Protokoll 6

Praktikum Technische Informatik

Tas Soti, Richard Wilhelm, Naja v. Schmude

24. Juli 2008

1 Aufgabe 13

1.1 Vorbereitung

In dieser Aufgabe geht es darum, über GPRS eine TCP Verbindung herzustellen, und so zwischen einer Webseite und dem Modul Daten auszutauschen, um die LEDs zu steuern bzw. den Status der Tasten anzuzeigen. Dabei stehen uns schon vorgefertigte Funktionen zu Verfügung, um eine GPRS und eine TCP-Verbindung aufzubauen. So wird z.B. durch die Funktion `hwp_gprsStartconnection` eine GPRS-Verbindung aufgebaut, die als Parameter eine Anzahl an Verbindungsversuchen und einen Handler erwartet. Sobald die GPRS-Verbindung steht, kann mittels `hwp_tcpStartActiveConnection` eine Verbindung an einen bestimmten Port einer bestimmten Adresse aufgebaut werden. Des weiteren muss auch hier die Anzahl der Verbindungsversuche und natürlich der Handler angegeben werden. Sobald auch diese Verbindung steht, kann dann per `hwp_tcpSendAsciiData` Strings verschickt werden. Dazu wird dann lediglich der String und die Anzahl der Zeichen übergeben.

1.2 Aufgaben und Durchführung

- *Es soll eine Verbindung mit der Seite hergestellt werden, über die dann die LEDs gesteuert werden können. Gleichzeitig soll bei jedem Tastendruck ein entsprechender Befehl an die Webseite geschickt werden.*

```
void a13_apply(void)
{
    a13io = adl_ioSubscribe(ADL_IO_Q25X1_GPO_0 |
        ADL_IO_Q25X1_GPO_3, 0, 0, 0, (adl_ioHdlr_f) NULL); //
        Lampe GPO_0 und GPO_3 anmelden

    wm_strcpy(a13_tcp_send_buffer, "SET:000000;");
```

```
//GPRS aktivieren
hwp_gprsStartConnection(5, a13_gprsHandler);

}

void a13_gprsHandler(s32 event, TeDHandle id)
{
    switch (event)
    {
        case ED_OK_GPRS_SESSION_SET:
            adl_atSendResponse(ADL_AT_UNUS, "//_GPRS_-_
                a13_gprsHandler:_[SUCCESS]_ED_OK_GPRS_SESSION_SET_\
                r\n");
            //TCP aktivieren
            hwp_tcpStartActiveConnection((u16)8007, "mackone.
                gotdns.com", 3, 3, a13_tcpHandler);
            break;

            // alle übrigen Fälle sind hier der Übersichtlichkeit
            wegen rausgenommen.
        }
    }

void a13_tcpHandler(s32 event, TeDHandle id, ascii *data,
    u16 length)
{
    ascii buffer1[12];
    ascii buffer2[120];
    ascii buffer3[20];

    switch (event)
    {
        case ED_INFO_WAITING_FOR_DATA: // dieser Zweig ist
            wichtig -> wir können hier unsere Daten schicken.
            adl_atSendResponse(ADL_AT_UNUS, "//_TCP_-_a13_tcpHandler
                :_[INFO]_ED_INFO_WAITING_FOR_DATA_\r\n");

            adl_atCmdCreate("AT+CMER=,1", FALSE, (
                adl_atRspHandler_t) NULL, NULL); // Aktivierung der
                Schalter
            adl_atUnSoSubscribe("+CKEV", (adl_atUnSoHandler_t)
                a13_tastenHandler); // Registrierung des Drückens
                von Tasten

            break;
    }
}
```

```

case HWP_TCP_DATA RECEIVED: // dieser Zweig ist auch
    wichtig, hier haben wir Daten empfangen und müssen
    die LEDs dementsprechend setzten.
    adl_atSendResponse(ADL_AT_UNUS, "//TCP- a13_tcpHandler
        : [INFO] HWP_TCP_DATA RECEIVED \r\n");
    wm_strncpy(buffer1, data, 11);
    buffer1[11] = '\0';
    wm_sprintf(buffer2, "Wir haben folgendes empfangen: %s
        \n\n", buffer1);
    adl_atSendResponse(ADL_AT_UNUS, buffer2);

    // Leds auslesen
    wm_strncpy(a13_tcp_send_buffer+4, buffer1 + 4, 1);
    wm_strncpy(a13_tcp_send_buffer+5, buffer1 + 5, 1);
    wm_sprintf(buffer3, "LED0: %c LED1: %c\n\n",
        a13_tcp_send_buffer[4], a13_tcp_send_buffer[5]);
    adl_atSendResponse(ADL_AT_UNUS, buffer3);

    // hier leds setzten
    if(a13_tcp_send_buffer[4] == '0') // Lampe 0
        ausschalten
        a13status0 = 0x00000000;
    else // Lampe 0 anschalten
        a13status0 = 0xFFFFFFFF;

    if(a13_tcp_send_buffer[5] == '0') // Lampe 3
        ausschalten
        a13status3 = 0x00000000;
    else // Lampe 3 anschalten
        a13status3 = 0xFFFFFFFF;

    adl_ioWrite(a13io, ADL_IO_Q25X1_GPO_3, a13status3); //
        neue Status schreiben
    adl_ioWrite(a13io, ADL_IO_Q25X1_GPO_0, a13status0);
    break;

    // auch hier wieder alle nicht benötigten Fälle
    weggelassen.
}

/* Tastendruck behandeln */
bool a13_tastenHandler(adl_atUnsolicited_t *parameter) {
    ascii tastennummer[2];
    ascii tastenstatus[2];
    ascii buffer[40];

```

```
wm_strGetParameterString(tastennummer, parameter->
    StrData, 1); // Taste auslesen
//adl_atSendResponse(ADL_AT_RSP, "\n\nTastennummer: ");
//adl_atSendResponse(ADL_AT_RSP, tastennummer);
wm_strGetParameterString(tastenstatus, parameter->
    StrData, 2); // Status auslesen
//adl_atSendResponse(ADL_AT_RSP, "\n\nTastenstatus: ");
//adl_atSendResponse(ADL_AT_RSP, tastenstatus);

if(wm_strcmp(tastenstatus, "0") == 0) // beim loslassen
    soll nix passieren
    return FALSE;

// Wir schauen, welche Taste gedrückt wurde, und setzten
// entsprechen die Werte
switch(tastennummer[0]) {
    case '0':
        adl_atSendResponse(ADL_AT_RSP, "Taste_ '0' gedrückt\n\n");
        wm_strcpy(a13_tcp_send_buffer+6, "1");
        wm_strcpy(a13_tcp_send_buffer+7, "0");
        wm_strcpy(a13_tcp_send_buffer+8, "0");
        wm_strcpy(a13_tcp_send_buffer+9, "0");
        break;
    case '1':
        adl_atSendResponse(ADL_AT_RSP, "Taste_ '1' gedrückt\n\n");
        wm_strcpy(a13_tcp_send_buffer+6, "0");
        wm_strcpy(a13_tcp_send_buffer+7, "1");
        wm_strcpy(a13_tcp_send_buffer+8, "0");
        wm_strcpy(a13_tcp_send_buffer+9, "0");
        break;
    case '2':
        adl_atSendResponse(ADL_AT_RSP, "Taste_ '2' gedrückt\n\n");
        wm_strcpy(a13_tcp_send_buffer+6, "0");
        wm_strcpy(a13_tcp_send_buffer+7, "0");
        wm_strcpy(a13_tcp_send_buffer+8, "1");
        wm_strcpy(a13_tcp_send_buffer+9, "0");
        break;
    case '3':
        adl_atSendResponse(ADL_AT_RSP, "Taste_ '3' gedrückt\n\n");
        wm_strcpy(a13_tcp_send_buffer+6, "0");
        wm_strcpy(a13_tcp_send_buffer+7, "0");
```

```
        wm_strcpy(a13_tcp_send_buffer+8,"0");
        wm_strcpy(a13_tcp_send_buffer+9,"1");
    break;
}

// Daten abschicken.
wm_sprintf(buffer,"Wir abschicken ab: %s\n\n",
    a13_tcp_send_buffer);
adl_atSendResponse(ADL_AT_RSP,buffer);

hwp_tcpSendAsciiData(a13_tcp_send_buffer, wm_strlen(
    a13_tcp_send_buffer)+1);

return FALSE;
}
```

Zunächst wird in der `a13_apply` die Anmeldung zu der IO-Funktionalität durchgeführt, wie das auch schon aus früheren Aufgaben bekannt ist. Zudem wird dann wie bereits weiter oben beschrieben die GPRS-Verbindung gestartet, der Handler, der sich dann um die Behandlung der GPRS-Ereignisse kümmert, ist die Funktion `a13_gprsHandler`. Zudem wird zu Beginn noch der `a13_tcp_send_buffer` initialisiert, so braucht man dann nicht immer den ganzen String neu schreiben sondern es ist später ausreichend einzelne Buchstaben direkt auszutauschen.

Wenn die GPRS-Verbindung steht, wird dann in der `a13_gprsHandler` das Event `ED_OK_GPRS_SESSION_SET` ausgelöst. Daher wird in diesem Fall dann wie auch bereits beschrieben per `hwp_tcpStartActiveconnection` eine TCP-Verbindung aufgebaut. Und zwar wollen wir uns zum Host `mackone.gotdns.com` am Port 8007 verbinden. An dieser Adresse befindet sich nämlich die Webseite. Der Handler für TCP-Events ist hierbei `a13_tcpHandler`.

Wenn jetzt auch die TCP-Verbindung erfolgreich hergestellt wird, wird das Ereignis `ED_INFO_WAITING_FOR_DATA` ausgelöst. In diesem Zustand können wir selber Daten abschicken. Daher registrieren wir zunächst einmal die Funktionalität der Tasten mit `AT+CMER=,1`, wie ja auch schon von früher bekannt ist. Damit wir dann auch das Tastendrücken registrieren, brauchen wir noch einen Handler, der auf `+CKEV` reagiert, in unserem Fall ist das die Funktion `a13_tastenHandler`. Da nur bei einem Tastendruck der Befehl übertragen werden soll, steckt diese Funktionalität im genannten Handler.

Im `a13_tastenHandler` müssen wir also zunächst auslesen, welche Taste gedrückt wurde, und ob es ein Tastendrücken oder das Loslassen einer Taste war. Wenn die Taste losgelassen wurde, können wir gleich die Methode verlassen, da wir dann nichts senden wollen. Andernfalls schreiben wir je nach dem welche Taste gedrückt wurde, eine 1 an die entsprechende Stelle im zu übertragenden Buffer. Dann übertragen wir den Buffer per `hwp_tcpSendAsciiData` und verlassen die Funktion.

Wenn wir hingegen in der Webseite LEDs schalten wollen und einen entsprechenden Befehl an das Modul übermitteln, wird im TCP-Handler das Ereignis HWP_TCP_DATARECEIVED ausgelöst und wir springen in den entsprechenden Zweig. Da die geschickten Daten, die sich im Feld data befinden, ohne Nullterminierung an uns übermittelt werden müssen, wir zunächst uns selber diese noch ranbauen. Dann können wir die entsprechenden Zeichen für die LEDs auslesen und entsprechend behandeln. Wenn wir eine Null bekommen, dann soll die entsprechende LED ausgeschaltet werden, d.h. wir setzen die Leitung komplett auf 0 mittels 0x00000000. Und für das Anschalten schreibt man dann 0xFFFFFFFF in die Leitung mittels adl_ioWrite.

Ein Testlauf des Programms kann wie folgt aussehen:

```
// Modul-Initialisierung: PIN-Eingabe abgeschlossen
// Modul-Initialisierung: Gerät funktionsbereit!
// GPRS - ed_DialupSetConfig: [SUCCESS]
// GPRS - ed_DNSSetConfig: [SUCCESS]
// GPRS - ed_GPRSSetConfig: [SUCCESS]
// GPRS - ed_DialupConnectionStart: [SUCCESS]
OK
OK
// GPRS - Callback: ED_OK_GPRS_SESSION_SET
// GPRS - a13_gprsHandler: [SUCCESS] ED_OK_GPRS_SESSION_SET
// TCP - ed_SocketSetConfig: [SUCCESS]
// TCP - ed_SocketTCPStart: [SUCCESS]
// TCP - Callback: ED_INFO_WAITING_FOR_DATA
// TCP - a13_tcpHandler: [INFO] ED_INFO_WAITING_FOR_DATA
[wir schicken von der Seite Daten]
// TCP - data request: maxLen = 536
// TCP - a13_tcpHandler: [INFO] HWP_TCP_DATARECEIVED
Wir haben folgendes empfangen: SET:1100000b
LED0: 1   LED1: 1
```

```
Taste '0' gedrückt
Wir schicken ab: SET:111000
// TCP - ed_SendDataExt: [SUCCESS]
[wir schicken Daten von der Seite ab]
// TCP - data request: maxLen = 525
// TCP - a13_tcpHandler: [INFO] HWP_TCP_DATARECEIVED
Wir haben folgendes empfangen: SET:0110000b
LED0: 0   LED1: 1
```

```
Taste '3' gedrückt
Wir schicken ab: SET:010001
// TCP - ed_SendDataExt: [SUCCESS]
// TCP - data request: maxLen = 525
```