H8-Menu

Klassifikation:	**Ausbildung**
Erstellt von:	**Benno Hübscher / Hanno Baumgartner**
Bearbeiter:	**Benno Hübscher**
Klasse:	**E99**
Ausgabe/Revision:	**1.0**
Datum:	**7. Dezember 2001**
Ausdruck:	**Freitag, 7. Dezember 2001 / 3:39**

Verteiler:

Name:	Adresse:	Tel.	
B. Hübscher	reselec ag	01/864 10 10	benno.huebscher@reselec .ch
H. Baumgartner	EVIS AG	01/908 11 11	h.baumgartner@gmx.net

Revisionsgeschichte:

Rev.	Revisionsgrund	Freigabe	
		Datum:	Visum:
1.0	Version 1.0 erstellt	07.12.01	BHU

Inhaltsverzeichnis:

Liste der Figuren

Liste der Tabellen

1 Allgemeine Angaben zum Projekt

1.1 Aufgabenstellung

Es soll ein Bedienoderfläche für das Hitachi H8-Entwicklungs Board realisiert werden. Das Menu kann über die Serielle Schnittstelle 1 am Hyperteminal sichtbar gemacht werden. Das H8-Menu soll als User-Interface in zukünftige Projekte eingebaut werden können. Damit andere laufende Processe nicht blokiert sind müssen die eingelesenen ASCI-Zeichen über eine interruptgesteuerte Schnittstelle verwaltet werden. Zur Demonstration des Geleisteten wird ein Menu erstellt welches die wichtigsten Funktionen des H8-Entwicklungs Boards steuert bzw. ausliest.

1.2 Übersicht *main();*

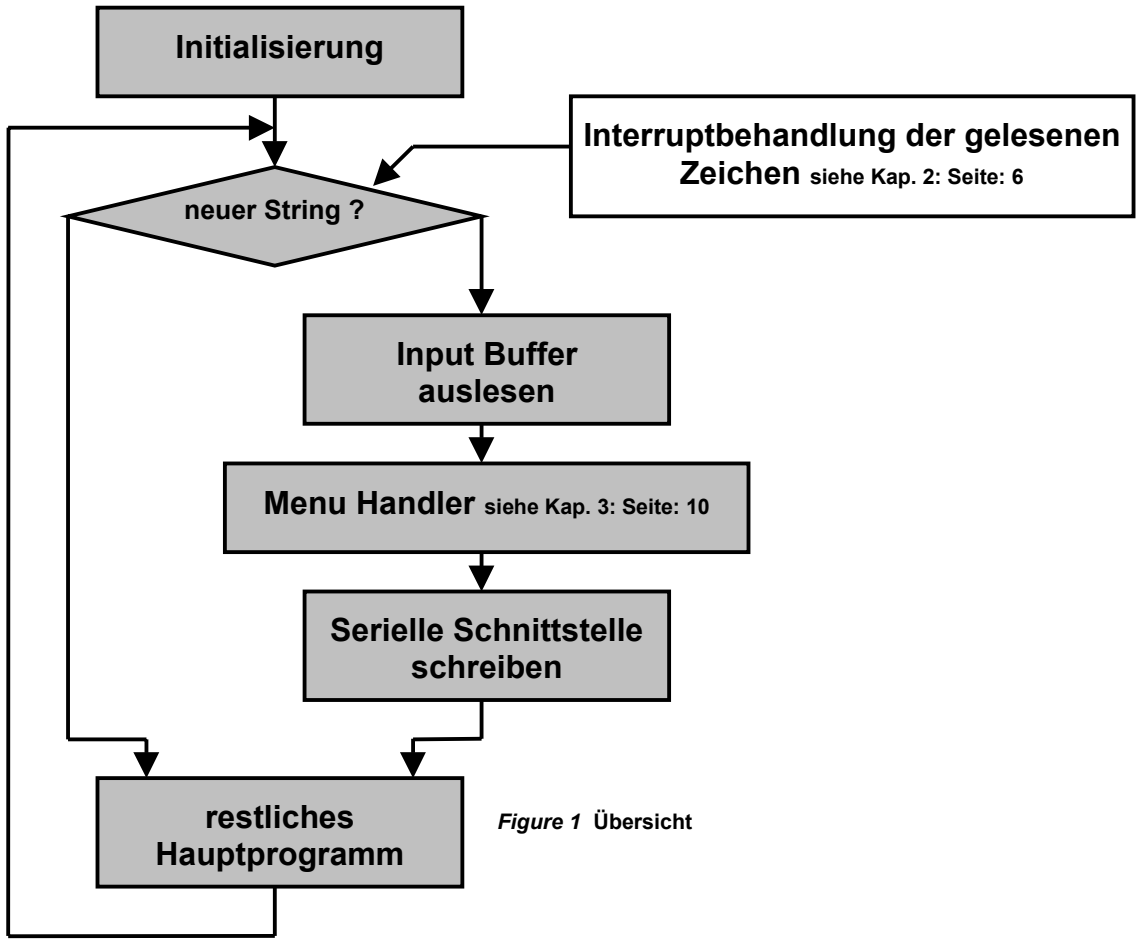

Figure 1 **Übersicht**

1.3 Ansicht in Terminalprogramm

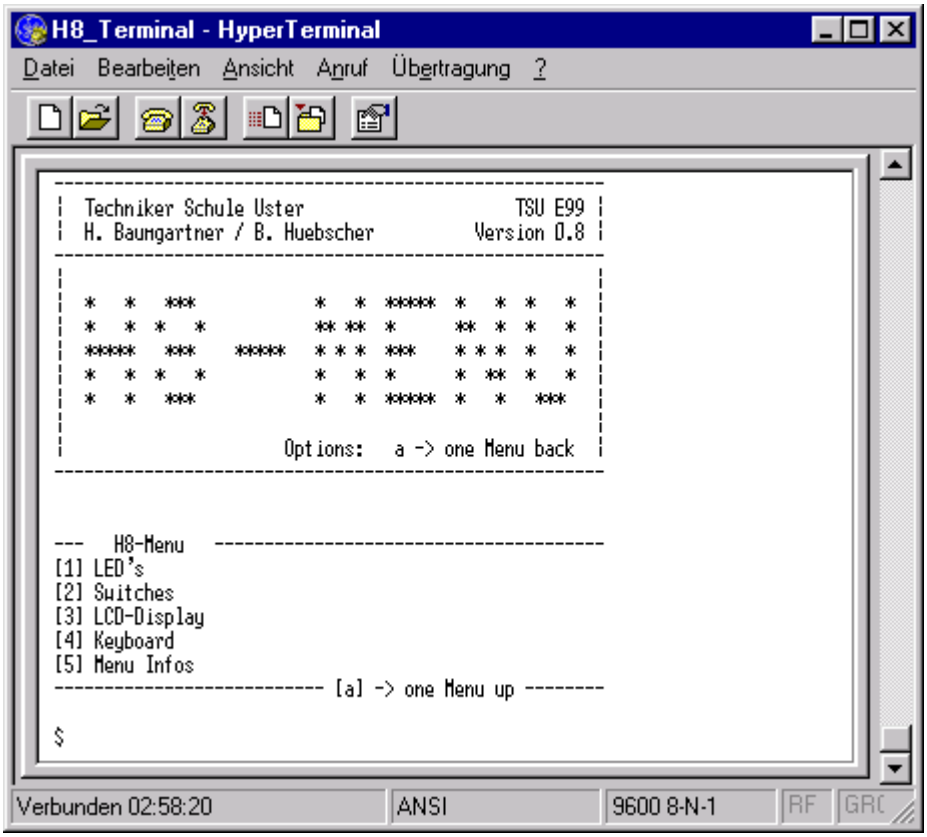

Figure 2 **Bildschirm im Terminalprogramm**

1.4 Schnittstelle zwischen Hauptprogramm und Menu

Es werden zwei globale Strings benötigt um zwischen dem Hauptprogramm und dem Menu zu kommunizieren. Die Eingabe erfolgt über `RS_IN_Buffer` und die Ausgabe über `RS_OUT_Buffer`. Die länge der Strings sind als Konstanten definiert: *`MAX_OUTPUT_STRING_LENGTH`* und *`MAX_INPUT_STRING_LENGTH`*

1.5 Aufteilung der Teilaufgaben

Pos.	Teilaufgabe	Bearbeitender	Besonderes / Probleme
1	Aufgabenstellung erarbeiten	H. Baumgartner / B. Hübscher	
2	Interruptgesteuerte Serielle Schnittstelle	H. Baumgartner	Initialisierung Register
3	Erstellung Menuvorlage / navigation innerhalb des Menus	B. Hübscher	State Machine / Speicherverbrauch
4	Funktionen zur Steuerung der Funktionen hinter den Menupunkten	H. Baumgartner / B. Hübscher	
5	Tests	H. Baumgartner / B. Hübscher	
6	Dokumentationen	H. Baumgartner / B. Hübscher	
7	Präsentation	H. Baumgartner / B. Hübscher	

Table 1 **Aufteilung Hauptaufgaben**

2 Interruptbehandlung der seriellen Schnittstelle

2.1 Beschreibung

Werden Zeichen an der Serielle Schnittstelle empfangen, so werden diese mittels Interrupt in ein Ringbuffer eingelesen. Da eine Eingabe im Menu immer mit der Returntaste bestätigt wird, werden alle Zeichen im Ringbuffer gespeichert, bis das Zeichen Carrige Return (0x0D) empfangen wird. Nun ist ein gültiger String vorhanden.

Um den Wert im Ringbuffer in einen String zu wandeln, wird das Carrige Return Zeichen durch das Zeichen `'\0'` ersetzt. Von jetzt an ist ein gültiger String in Ringbuffer vorhanden.

Sollt ein Übertragungsfehler wie: Parity-, Framing- oder Overrun- Error auftreten, so werden diese durch eine eigene Interruptroutine auf dem Display angezeigt.

Achtung: Das Flashprogramm, welches sich auf dem TSU Board befindet, weißt einen Fehler auf. Die Aktivierung eines RS232 Empfangsinterrupt ist nicht möglich. Für die Funktionalität dieser Funktion, ist es zwingend notwendig, die nächst ältere Version des Flashprogrammes herunterzuladen.

2.2 Registerinitialisierung

Die Initialisierung wurde gemäss Flussdiagramm im Hardware Manual (Seite 497) für die Com 1 Schnittstelle vorgenommen.

- **Start der Initialisierung**
- **Löschen der TE- und RE- Bit's im Serial Control Register**
- **Auswählen des Clocksignals im Serial Control Register**
- **Kommunikationsformat im Serial Mode Register einstellen**
  => asynchron, 8 Data Bit's, keine Parität, 1 Stop Bit
- **Faktor für Baudrate in Bit Rate Register laden => 0x3B für eine Baudrate von 9600bit/s**
- **Warten für 1Bit Interval (Zeitdauer die benötig wird, um 1 Bit zu senden oder zu empfangen)**
- **Setzen des Interrupts, sowie der TE und RE Bit's.**
- **Ende der Initialisierung**

2.2.1 Ringbuffer

Um Daten interrupt gesteuert einlesen zu können, wird ein einfacher „Ringbuffer" geschaffen. Somit wird verhindert, dass der Empfangsbuffer überläuft.

Werden aus irgendwelchen Gründen zu viele Zeichen vom Terminal PC her übertragen, so werden die älteren Daten im „Ringbuffer" immer wieder mit den neusten Zeichen überschrieben. Nachdem ein vollständiger String aus dem Ringbuffer gelesen wurde, wird der nächste String wieder von der Position [0] an eingelesen.

Sollte also die grösse des Buffers für längere Befehle nicht ausreichen, so muss nur die Konstante: `MAX_INPUT_STRING_LENGTH` erhöht werden.

Init:

```
char    RS_Daten[MAX_INPUT_STRING_LENGTH];
```

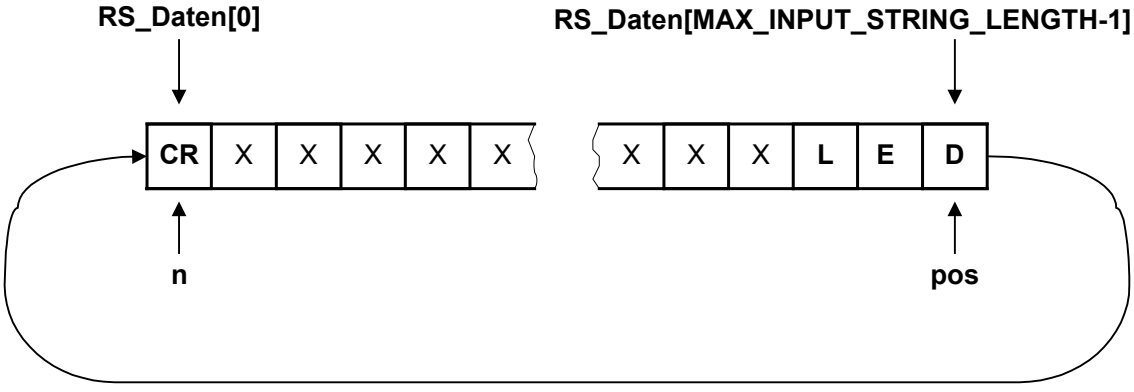

Figure 3 **Ringbuffer**

2.3 Interrupt Routinen

2.3.1 Empfangen der RS232 Daten

2.3.1.1 Beschreibung

Empfangene Daten werden nur ins „Ringbuffer“ geschrieben, wenn noch kein gültiger String vorhanden ist. Somit wird verhindert, dass ungelesene Daten überschrieben werden.

Ein String soll dann vollständig sein, wenn der User am Terminal Programm die `<Enter>`- Taste betätigt. Somit muss das Enter- Zeichen im String durch das Abschlusszeichen `'\0'` ersetzt werden.

2.3.1.2 Struktogramm

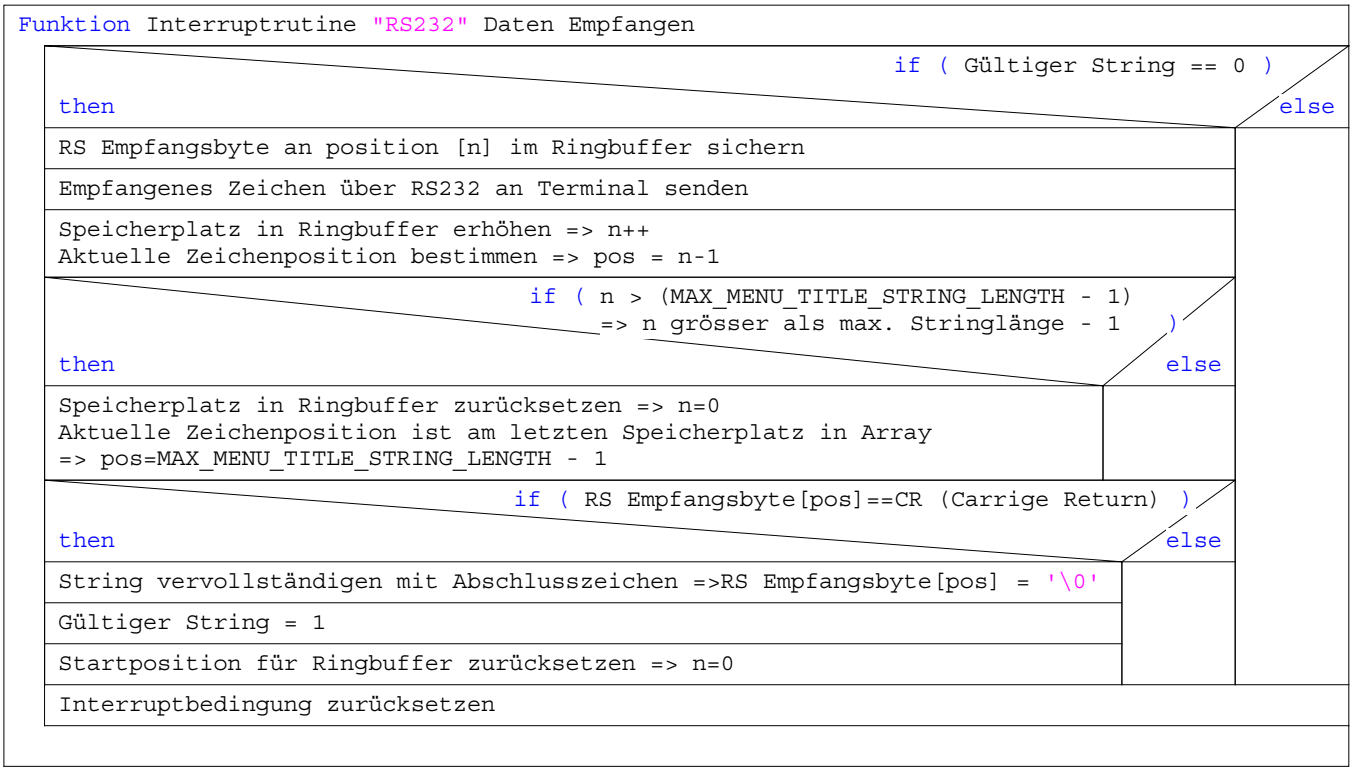

Figure 4 **Struktogramm: Empfang RS232-Daten**

2.3.2 Auswerten von RS232 Errormeldungen

2.3.2.1 Beschreibung

RS232 Errormeldungen wie: Overrun- Framing- und Parity- Error sind im Serial Status Register enthalten. Die entsprechenden Bit's werden ausmaskiert und entsprechend Ihrer Bedeutung auf dem Display angezeigt.

2.3.2.2 Struktogramm

Figure 5 **Struktogramm: RS232-Errors**

2.4 Sichern der ISR eingelesenen Daten

2.4.1 Beschreibung

Indem die ISR eingelesenen Daten gesichert werden, kann das serielle Empfangen von Daten noch vor dem vollständigen Ablaufen des Hauptprogrammes wieder zugelassen werden. Die empfangenen Daten werden aus dem Ringbuffer in den `RS_IN_Buffer` kopiert. Somit kann nach dem Ausführen dieser Funktion der Befehl: `Valid_String=0` ausgeführt werden. Mit diesem Befehl wird das einlesen von neuen seriellen Daten ermöglicht.

2.4.2 Struktogramm

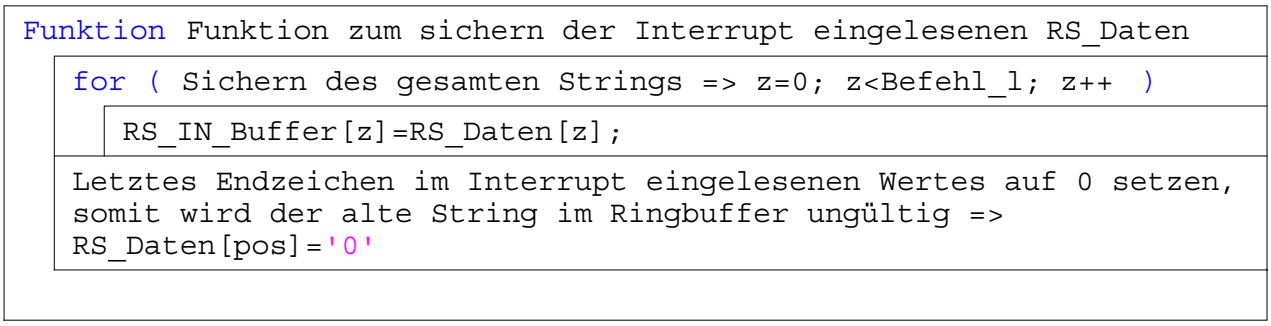

Figure 6 **Struktogramm: Sichern der Eingabe**

Unmittelbar nach dem Ausführen dieser Funktion kann der Befehl: `Valid_String = 0;` ausgeführt werden. Dies hat dann zur Folge, dass wieder seriell empfangene Daten eingelesen werden.

In unser Anwendung erfolgt die Freigabe für das Einlesen der seriellen Daten unmittelbar nach der Menuausgabe.

3 Menu Handler

3.1 Beschreibung

Der MenuHandler nimmt einen String (`RS_IN_Buffer`) auf, wertet ihn aus und reagiert entsprechend. Falls Ausgaben auf die serielle Schnittstelle erfolgen sollen werden sie über den Ausgangs-String (`RS_OUT_Buffer`) zum Terminal gesendet. Die Navigation der Position im Menu wird mit einer *StateMachine* gesteuert.

3.2 State Machine

Jeder State ruft eine ihm zugeordnete Funktion auf. Zum Beispiel ruft der State `ST_MENU_1` die Funktion `Menu_do_Menu_1( &menu_state, RS_OUT_Buffer );` auf. Sie gibt einerseits den nächsten MenuState (`menu_state`) und andernseits den zum Terminal (über RS-Schnittstelle) zu sendenden String (`RS_OUT_Buffer`) zurück.

Die Position innerhalb der StateMachine wird mit einer globale Aufzählungsvariablen (Typ: `enum` Name: `enum_states`) definiert. Diese wird pro state je nach Eingabe (`RS_IN_Buffer`) neu gesetzt und bestimmt somit beim folgenden Durchlauf durch die Routine die nächste Position (state). Mit der Eingabe <a> wird der übergeordnete state; also das übergeortnete Menu aufgerufen.

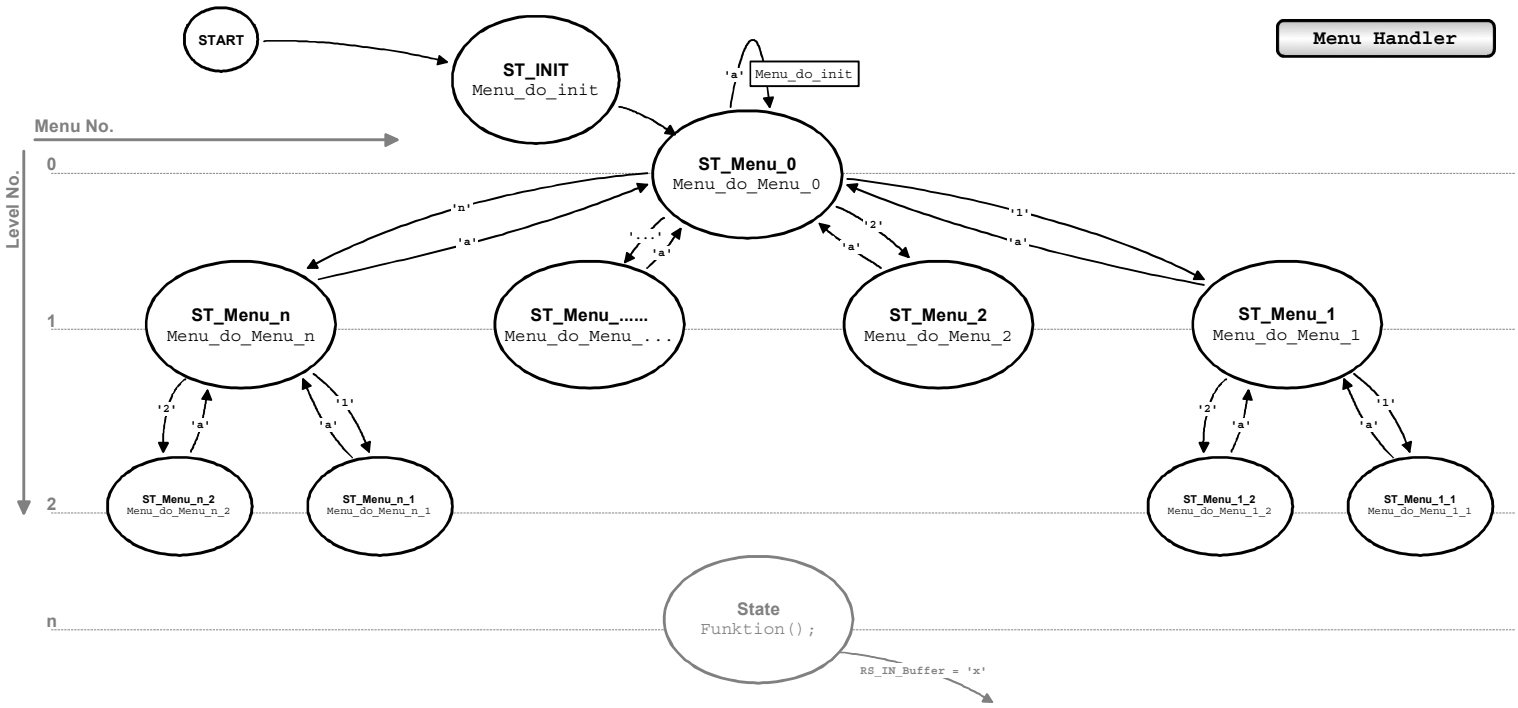

Figure 7 **StateDiagramm MenuHandler**

3.3 Menu Funktionen (Menu_do_Menu_xy)

Innerhalb dieser Funktion wird die Benutzereingabe (`RS_IN_Buffer`) ausgewertet und bei einer gültigen Eingabe die entsprechende Funktion aufgerufen und/oder der folgende state definiert.

3.4 Ausgabe Schnittstelle

Zur Ausgabe wird die Standardfunktion `CBTSU_RSWriteLn( RS_OUT_Buffer );` aus der CBTSU2 - Bibliotek benützt.

3.5 Erweiterung des Menus

Das in diesem Projekt erstellte Menu (H8_Menu) ist natürlich nur eine Vorlage für künftige Microcontroller-Applikationen und muss somit angepasst werden.

3.5.1 Namen und Texte

Alle im Terminal (z.B. Hyperterminal) ersichtlichen Menunamen sind in einem zentralen Datenfeld (`MenuDefs`) gespeichert. Ein Feld vom Typ `TMenu` beinhaltet die Ebene (`LevelNo`), den Menunummer und -übertitel (`MenuNo`, `MenuText`) und die einzelnen Menupunkte (`PointNo`, `PointText`). Die Anzahl solcher Felder wird aufgrund Speicherknappheit durch eine Konstante begrenzt. Es entsteht eine Tabelle die einfach editiert werden kann:

```
typedef struct SMenu {
     int  LevelNo;
     int  MenuNo;
     char MenuText [ 14 ];    // max 13. char per Menuname
     int  PointNo;
     char PointText[ 16 ];    // max 15. char per Pointname
} TMenu;

#define TOTAL_MENU_Points 60

TMenu MenuDefs[TOTAL_MENU_Points]= {

// only the first Level- and Menu-Names are important !!!
// ____________________________________________________
// Level|Menu|   MenuTitle   |Point|     PointName      |
//      |    |               |     |                    |
     { 1,  0, "    H8-Menu   ",  1, "LED's           "},
     { 1,  0, "              ",  2, "Switches        "},
     { 1,  0, "              ",  3, "LCD-Display     "},
     { 1,  0, "              ",  4, "Keyboard        "},
     { 1,  0, "              ",  5, "Menu Infos      "},
     { 2,  1, "    LED's     ",  1, "LED's green     "},
     { 2,  1, "              ",  2, "LED's red       "},
     { 3, 11, " LED's green  ",  1, "LED green 0     "},
     { 3, 11, "              ",  2, "LED green 1     "},
.... u.s.w.
     { 0,  0, "no Menu       ",  1, "DummyMenuPoint "},
//      |    |               |     |                    |
// Level|Menu|   MenuTitle   |Point|     PointName      |
// ----------------------------------------------------
```

Die Variablen LevelNo. MenuNo und PointNo dienen nur zum effektiven Auslesen (`Menu_print_by_Menu();` , `Menu_getMenuTitle();`) aus dem Datenfeld.

3.5.2 Positionen

Zusätzlich muss die StateMachine (`enum_states`, `MenuHandler();`) und deren Funktionsaufrufe (`Menu_do_menu_`xy) dem neuen Menu entsprechen.

4 Eingebaute Board-Testfunktionen

4.1 LED's

4.1.1 Beschreibung

Der Funktion: Anzeige der roten oder grünen LED, wird eine Steuervariabel übergeben. Der Wert in dieser Variabel sollte ein ASCI Zeichen von 0 bis 8 enthalten. Dies wird geprüft und anhand des entsprechenden Wertes werden die LED's auf dem Board angesteuert.

Steuervariabel	LED Anzeige
'0'	alle LED's aus
'1'	1. LED an, alle anderen aus
'2'	2. LED an, alle anderen aus
'3'	3. LED an, alle anderen aus
'4'	4. LED an, alle anderen aus
'5'	5. LED an, alle anderen aus
'6'	6. LED an, alle anderen aus
'7'	7. LED an, alle anderen aus
'8'	8. LED an, alle anderen aus

4.1.2 Struktogramm

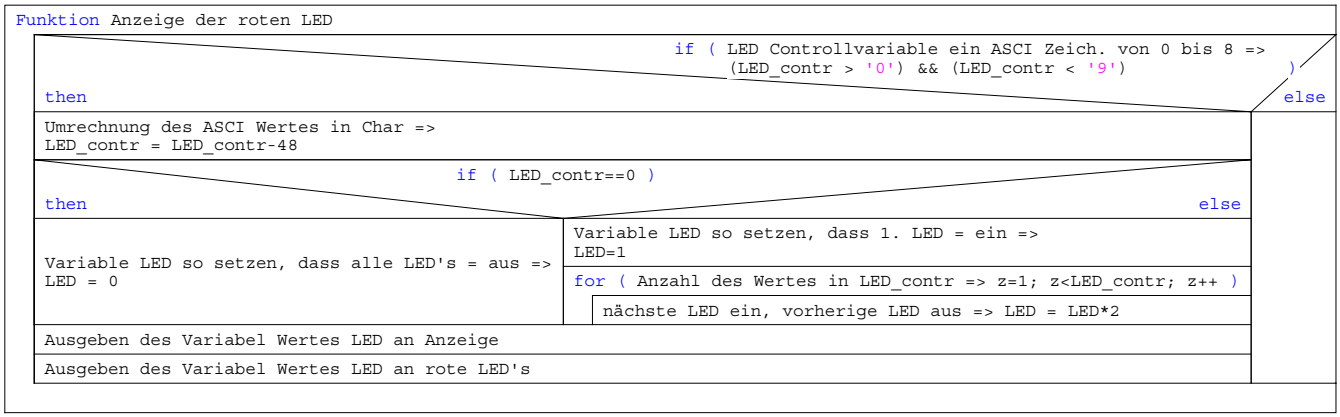

Figure 8 **Struktogramm: LED's**

4.2 Schalter

4.2.1 Beschreibung

Der Funktion: Anzeigen der Schalterposition im HEX- DEZ- oder BIN- Format, wird eine Steuervariabel mitgegeben. Anhand dieses Wertes wird dann bestimmt, in welchem Format der Wert der Schalter dargestellt werden soll.

Sollte der Wert im Binärformat angezeigt werden, so müssen die Schalterwerte umgerechnet werden.

4.2.2 Struktogram

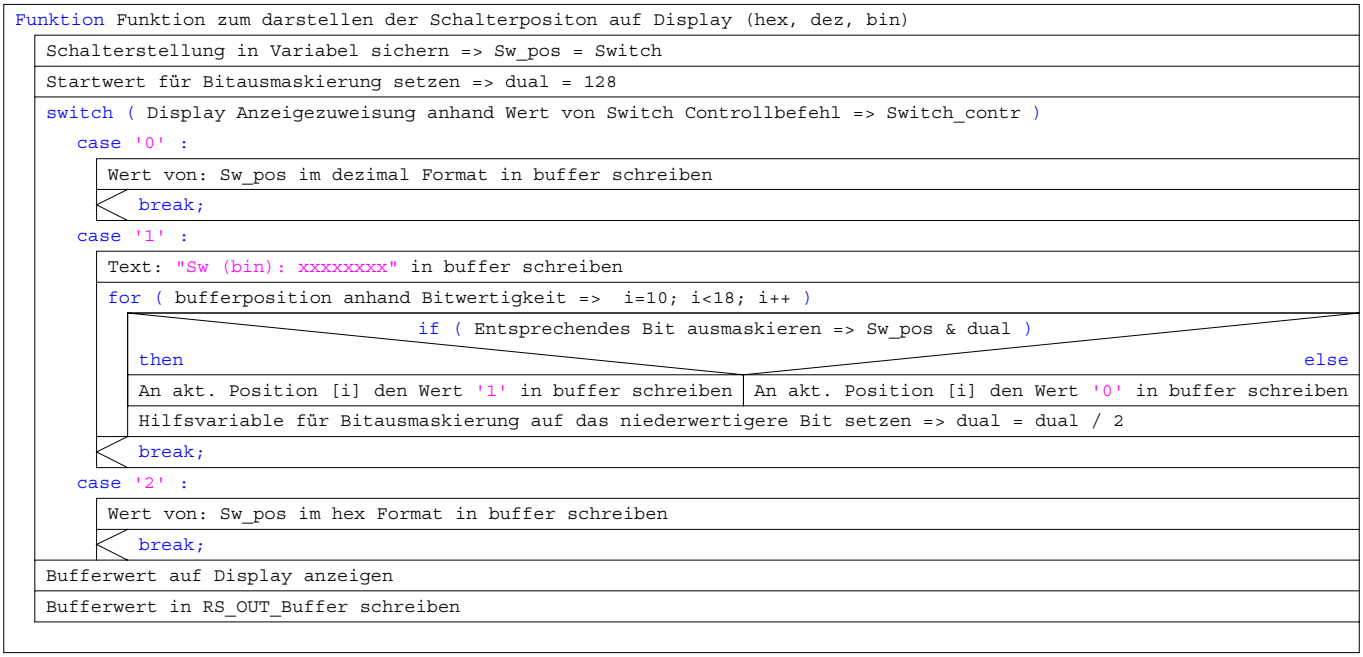

Figure 9 **Struktogramm: Schalter**

4.3 Keyboard

4.3.1 Beschreibung

Ist eine Tastatur am Board angeschlossen, so kann die gedrückte Taste auf dem Display angezeigt werden. Das Einlesen der gedrückten Taste erfolgt mittels Interrupt. Die Taste muss also vor dem anwählen des Menupunktes: „Display Keys“ gedrückt werden.

Im Interrupt wird der Tastaturwert eingelesen. Von diesem Wert wird noch ein Offset von 0x0F abgezogen. Da es sich hier um eine Tastatur mit Ziffern von 0-9 und Buchstaben von A-F handelt, muss der Tastaturwert einzeln dem beschrifteten Wert zugewiesen werden, was anhand einer Case- Anweisung erfolgt.

4.3.2 Initialisierung

Um den Externen Interrupt für die Tastatur einzuschalten, muss der Befehl: IER = 0x01; in der Hardwareinitialisierung eingefügt werden.

4.3.3 Struktogramm

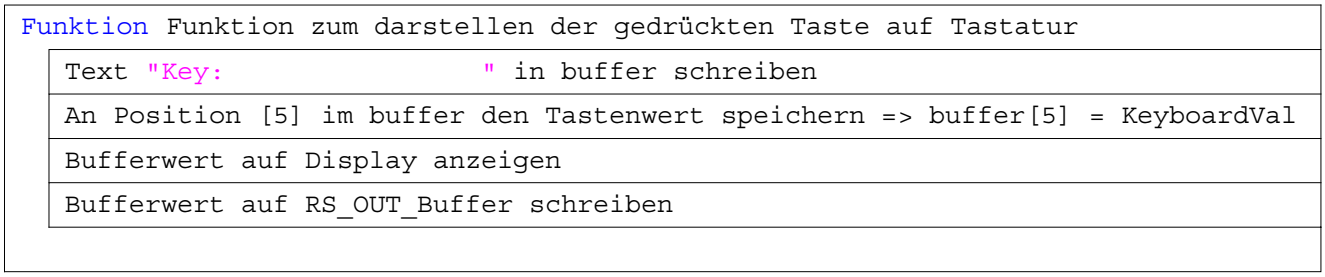

Figure 10 **Struktogramm: Keyboard**

5 Anwendung des H8-Menu

5.1 Hyperterminal

Das Hyperterminal kann in seiner Stanardkonfiguration (**9600**Baud, **8**dataBits, **no**Parity, **1** StopBit, noHW-Protokoll) betrieben werden. Für die ASCI-Zeichen Versand ist keine spezielle Einstellung erforderlich.

6 Analyse

6.1 Beurteilung Endresultat

Nach anfänglichen Schwierigkeiten ist doch noch ein brauchbares Endresultat entstanden. Der falsche Debugger Code im Flasch-Speicher des Entwicklungsboards und das Fehlverhalten der einfachen Funktion `strcpy(string, "");` verbrauchten sehr viel Zeit und Energie, die Am Schluss eher für zusätzliche Menufunktionen hätten eingesetzt werden können.

6.2 Probleme

6.2.1 RS-Schnitstelle

Der BootCode auf dem H8-TSU Entwicklungs Board neu geladen werden, da die Interrupt gesteuerte Serielle Schnittstelle trotz grossen Anstrengungen nicht funktionierte.

6.2.2 MenuHandler

Das kopieren eines leeren Strings (`strcpy(string, "")`) wird vom CYGNUS-Kompiler nicht richtig übersetzt. Alle diese Ausdrücke wurden duch (`string[0] = '\0'`) ersetzt.

6.3 Verbesserungsvorschläge

Der Aufbau des Menus würde mit einem Baum (verkettete Liste) flexibler und Speichersparender realisierbar.

7 Anhang

7.1 SoureCode