

DXM OPENHANDHELD SOURCECODE

© Diamex 2009

Wrtten by Erwin Reuss, ER-Tronik

Diese Software dient zur Demonstration der Ansteuerung des DXM1 und die Auswertung der OBD2 Daten. Die Software darf zum privaten Gebrauch benutzt und verändert werden. Eine kommerzielle Verwendung der kompletten Software oder Teile davon ist untersagt.

Ich weiß, jeder Programmierer hat einen eigenen Stil seinen Sourcecode zu formatieren und zu dokumentieren. Der vorliegende Openhandheld Sourcecode wurde von mir in relativ kurzer Zeit geschrieben, da ich für den ursprünglich vorgesehenen Programmierer einspringen musste. Aus diesem Grund hatte ich nur wenig Zeit, die Funktionen ausführlich zu dokumentieren. Auch wären sicher noch einige Funktionen mehr möglich gewesen oder das Grafikdisplay durch Animationen besser in Szene gesetzt worden, aber das ist eben das Ziel eines Opensource Betriebssystems. Macht was draus, wenn es Euch nicht gefällt. Ich denke jedoch, wer sich intensiv mit der Materie beschäftigt, wird auch mit meinem oft undokumentierten Code einigermaßen klar kommen.

Der Openhandheld-Sourcecode für den AT90CAN128 wurde in C geschrieben und mit WinAVR kompiliert. Wenn das WinAVR-Paket auf dem Rechner installiert ist, sollte der Aufruf von „make“ im Hauptverzeichnis des Sourcecode den kompletten Code fehlerfrei erstellen. Das hierbei erzeugte Hexfile „_hhopen.hex“ kann mit den meisten AVR-Programmieradapttern in den CAN128 einprogrammiert werden. Es wird kein Bootloader benötigt, die Software beginnt ab Adresse 0x0000. Die Einstellung der Fuses befindet sich am Anfang der Datei „main.c“. Es wird kein EEPROM-File benötigt, das interne EEPROM des Controllers wird beim ersten Start nach der Programmierung automatisch aktualisiert.

Bitte darauf achten, dass der Datenbereich die 64 kByte Grenze nicht überschreitet. Die Routinen sind allesamt nur für den unteren Speicherbereich ausgelegt, es gibt keine FAR-Aufrufe, da das Projekt eigentlich für den AT90CAN32 geplant war, aber wegen des knappen Flash- und RAM-Speichers haben wir uns dann doch für den größeren AT90CAN128 entschieden.

Verbesserungen und Funktionserweiterungen:

- Ich habe die Werte der Sensordaten nicht gespeichert, also müssen immer alle Werte neu gelesen werden, wenn man die Liste scrollt. Dies hat natürlich damit zu tun, dass der RAM-Speicher ziemlich knapp bemessen ist.
- Ebenso werden manche PIDs doppelt gelesen und einige müssten überhaupt nur einmal gelesen werden (z.B. 0x1C OBDSUP), da sich diese nie verändern. Gleiches gilt für die Sensordaten in den Freeze Frames.
- Das letzte verwendete Protokoll wird nicht gespeichert, so muss man jedes mal wieder das gewünschte Protokoll auswählen oder einen Autoscan durchführen.
- Für die Lambdasondenwerte könnte man einen eigenen Menüpunkt einbauen, damit die Gasumrüster alle Werte auf einem Blick sehen.
- Mode 7, also die während der aktuellen Fahrt anstehenden Fehlercodes könnten auch noch ausgelesen und angezeigt werden.
- Erweiterung der PID-Tabelle über \$40 hinaus.
- Bisher wurden nur metrische Ausgaben berücksichtigt, wer statt km/h mph oder statt °C °F bevorzugt, muß die entsprechenden Umrechnungen in der Datei „pid_disp.c“ vornehmen.
- Natürlich muß auch die Fehlercodeliste erweitert werden (in dtc_data.c). Aus rechtlichen Gründen habe ich nur ein paar Codes integriert um zu zeigen, wie man das beliebig erweitern kann. Hier bitte auch darauf achten, dass der gesamte Code inklusive Daten in den unteren Speicherbereich bis 64kB passt, da sonst alle Routinen auf FAR-Adressierung umgestellt werden müssen.

HINWEIS:

Die Zeichensätze für das Display wurden mit dem Freeware Font-Editor von H.Reddmann erstellt. Dieses Programm befindet sich zusammen mit den 3 benutzten Fonts im Verzeichnis /font.

FAQ:

Warum eigentlich ein AT90CAN128?

Es kann natürlich auch ein ATmega128L eingesetzt werden, der ist absolut pin-kompatibel zum AT90CAN128 und bis auf die CAN-Schnittstelle hardwaremäßig identisch. Da wir jedoch den AT90CAN128 in anderen Projekten einsetzen, können wir auf einen großen Lagerbestand zugreifen so dass es keinen Lieferengpass gibt.

Warum sind auf der Platine einige Teile unbestückt?

Es handelt sich hier um Bauteile, die für spätere Erweiterungen vorgesehen sind. Diese Teile werden in der aktuellen Software nicht angesprochen, hier ist eure Kreativität gefragt. Wer möchte, kann die Teile selber nachrüsten und die Software anpassen.

Diese sind im Einzelnen:

- Echtzeituhr RTC8564 (Epson) mit I2C-Schnittstelle. Hier müssen noch die I2C-Routinen für den AVR geschrieben werden.
- Dataflash AT45DB161 (Atmel) mit SPI-Schnittstelle. Die SPI-Routinen sind schon für das Display vorhanden, lediglich der Chip-Select muß für den Flash-Speicher programmiert werden.
- AT90USB162 (Atmel) mit CDC-Software als USB/Seriell-Wandler. Hier hätte genauso gut ein USB/Seriell-Wandler-Chip von FTDI eingesetzt werden können. Da diese Chips jedoch relativ teuer sind und vieles von den Funktionen nicht benötigt wird, ist die Lösung mit dem USB-Controller einfacher.

Die Definitionen der Portpins für die Erweiterungschips sind schon in „system.h“ vorhanden.

Wofür ist die 20-polige Stiftleiste vorgesehen?

Der komplette Port F des AT90CAN128 befindet sich auf dieser Leiste. Er kann sowohl als digitale Ein- oder Ausgänge programmiert werden oder als Analogeingänge zum Messen von Spannungen (Achtung! Max. 2.56 Volt). Des Weiteren liegen hier die CAN-Leitungen TX und RX sowie die Spannungen 3,3V und 5V und Masse an. Die restlichen Pins werden zum Test und Update des DXM-Moduls bei der Herstellung benötigt.

Warum läuft der Controller nur mit 8 MHz?

Die Spezifikation in den Datenblättern von Atmel lassen eine höhere Taktfrequenz (also 16MHz) nur bei Betriebsspannungen ab 4,5V zu. Da der Controller im Openhandheld nur mit 3,3V betrieben wird, haben wir uns für einen 8 MHz Quarz entschieden.

Die Schaltung wird sehr warm, ist das normal?

Ohne Hintergrundbeleuchtung würde die gesamte Schaltung nur lauwarm werden. Die RGB-Power-LEDs erzeugen jedoch eine ordentliche Temperatur, die aber absolut unkritisch ist.

Was ist im Verzeichnis PROG?

Hier befindet sich ein kleines Tool „HHOpenProg.exe“ speziell für den AVR-ISP-Programmer der Firma Stange Distribution. Das grafische Tool ist nur für den AT90CAN128 ausgelegt und spricht mit dem Programmer über „avrdude“, das sich im selben Verzeichnis befinden muß. Wichtig ist auch, dass sich die Datei „libusb0.dll“ im selben Verzeichnis befindet, obwohl sie eigentlich nicht benötigt wird.

HISTORY:

Update v1.2.1:

Neu:

Bluetooth-Mode integriert. Funktioniert natürlich nur mit optionalem BTM-222 Bluetooth-Modul auf dem Expansionsbus. Interne Übertragungsrate ist 19200 Baud.

ACHTUNG!!! Bitte mit Bluetooth nicht die DXM-Parameter permanent verstellen (z.B. ATM1 und danach die Baudrate, Echo, Linefeed usw. speichern). Wenn dies trotzdem passiert ist, kann das HHOPEN-Bios nicht mehr gestartet werden. In diesem Fall muß das DXM resettet werden, hierzu PIN30 des DXM mit Masse verbinden und Stromversorgung anlegen. Danach sollte alles wieder wie gewohnt funktionieren.

Update v1.1.2:

Bugfix:

Nach Aufruf des Freeze-Frame-Menüs stimmte die PID-Liste im Sensor-Menü nicht mehr. Dies wurde dadurch beseitigt, dass das Sensor-Menü nach Rückkehr aus dem Freeze-Frame-Menü wieder neu initialisiert wird.

Update v1.1.0:

Bugfix:

Im Automatic-Scan-Modus wurde häufig KWP2000-Fast-Init nicht gefunden. Dies wurde durch Änderung des Scan-timings geändert.

Bugfix:

Wenn das Steuergerät kein Modus 9 (Fahrgestellnummer auslesen) unterstützt hat, hat sich das Bios in eine Endlosschleife aufgehängt. Display zeigte 0900 und blieb stehen.