Dokumentation der GameBoy Software des Projekts

GBDSO

(Gameboy als digitales Speicher Oszilloskop)

Inhalt:

1.) Hardware Grundlagen, Einschränkungen
2.) Die verschiedenen Untermenüs
3.) Programm-Code

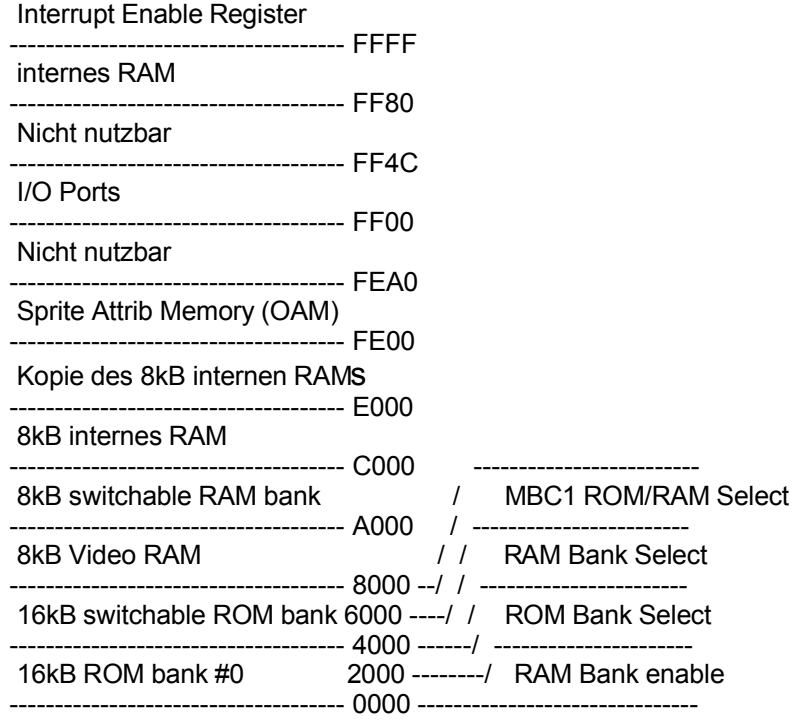
1.: Hardware Grundlagen

Basis unseres Projektes ist ein Nintendo Gameboy Classic mit folgenden für uns wichtigen Eigenschaften:

- CPU: 8bit Z80-ähnlicher Prozessor mit 4.2 Mhz
- 8bit Datenbus, 16bit Adressbus
- 8kB internes RAM, Adressbereich für 8kB externes RAM
- 8kB internes Video RAM
- 32kB reservierter Adressraum für externes ROM
- LCD Display mit 160*144 Pixeln, 4 Graustufen
- 4 Eingabetasten und Steuerkreuz

Die Software wurde in C implementiert, und mit Hilfe des gbdk-2.95 Compilers (http://gbdk.sourceforge.net/) in ein für die Gameboy Architektur ausführbares ROM Image compiliert. Im GBDK (Gameboy Development Kit) sind bereits Funktionen vorhanden um auf die spezielle Gameboy Hardware zuzugreifen, z.B. Joypad, Bildschirm Sound). Die Tests unter Windows wurden mit dem Emulator NO$GMG (Nocash Gameboy Emulator http://www.work.de/nocash/) durchgeführt. Enthalten ist auch ein guter Debugger mit direktem Zugriff auf alle Adressen.
Die Beschränkung auf 32kB Programmgröße im externen ROM kann durch einblenden von anderen ROM-Bänken in die 16 oberen kB des Adressraumes umgangen werden. Dies war bei uns allerdings nicht nötig, da das Programm nur ca 20kB Speicher benötigt.

Gameboy Memory Map:



Beim Compilieren des ROM Images müssen bestimmte Carakteristika der Boot-Sequenz des Gameboys berücksichtigt werden. Beim einschalten werden 256 Byte von Adresse 0x0 ab eingelesen. Dann wird von $104 bis $133 das Nintendo Logo für die Startanimation geladen. Wenn diese Bitfolge nicht mit der intern gespeicherten Originalversion übereinstimmt wird der Bootvorgang abgebrochen (eine Art Kopierschutz). Des weiteren werden hier Einstellungen vorgenommen bez. der Eigenschaften des ext. RAM/ROMs.
Die Funktionen des GBDSO Programmes sind folgende:

- Daten der Kanäle 1 und 2 von 0xA001 und 0xA002, je signed int, 8Bit ständig einlesen.
- Die Arrays mit den Daten skalieren und auf die gegebenheiten des Display anpassen.
- Die Daten graphisch ausgeben.

Gleichzeitig muss das Programm auf Benutzereingaben warten, die der Benutzer mit den Tasten eigibt. Mögliche Eingaben sind:

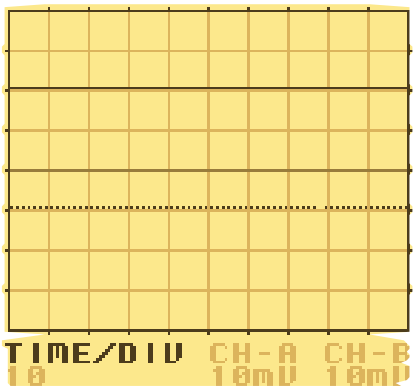
- Modus des Kanals A/B setzen auf: Ground, AC, DC
- Amplitude (Volt/DIV) eines Kanals setzen (Skalierung der Y-Achse)
- TIME/DIV setzen (Skalierung der X-Achse)

Je nach Einstellung muss der Gameboy ein Signal an den externen Mikrocontroller absetzen, damit dieser den A/D-Wandler demensprechend einstellen kann.

Probleme:

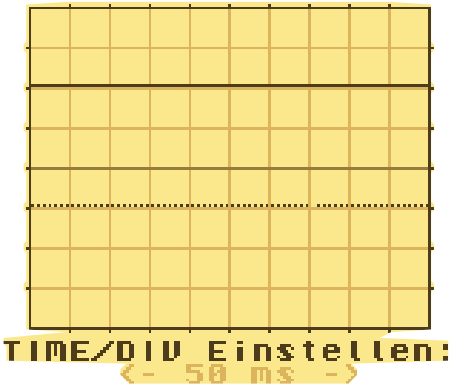
- Da sämtliche Abfragen und Vorgänge in while-Schleifen ablaufen ist das Programm relativ langsam, und ich habe einige Änderungen auf Kosten der Genauigkeit eingebaut um Performance zu gewinnen.
- Der verwendete GB-Compiler kennt zwar die Datentypen long und float, kann aber keine Berechnungen damit durchführen, deshalb basiert der gesamte Code auf 8Bit int Werten.

2.: Die verschieden Untermenüs:



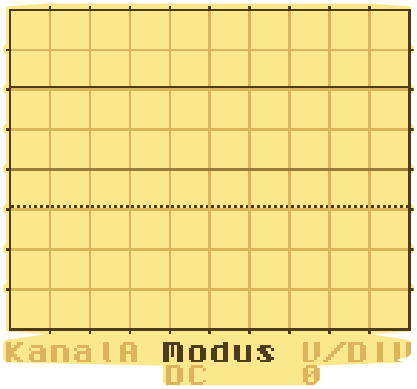
Hauptmenü:

Zwischen den 3 Menüpunkten TIME/DIV, CH-A und CH-B wird mit den Pfeiltasten navigiert. Auswählen mit <<SET>> (Emulator: l.Strg), Rückkehr zum Hautmenü mit <<RESET>> (Emulator: <<RETURN>>).
Wird ein Menüpunkt ausgewählt, so erscheint das entsprechende Untermenü.



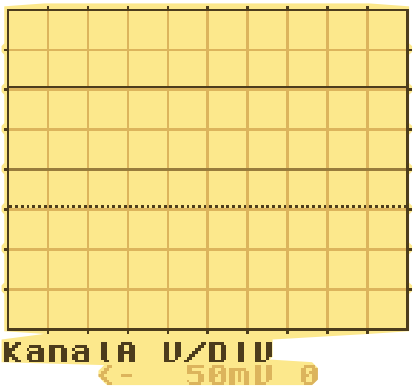
TIME/DIV Menü:

Hier wird die Zeit eingestellt, der 10 Kästchen in X-Richtung entsprechen. Mögliche Werte: 1ms bis 100 ms



Kanaleinstellungen Menü:

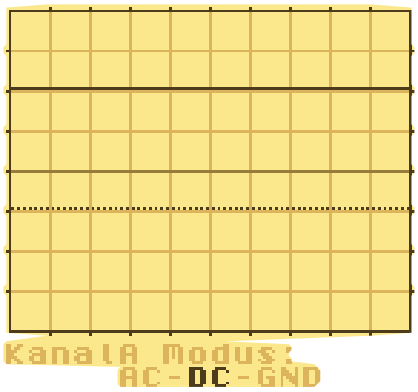
Auswahl bei bei gewähltem Kanal, ob der Modus oder die Amplitude geändert werden soll.



Amplituden-Menü:

V/DIV für vorher selektierten Kanal einstellen. Mögliche Werte:

10 mV/Div, 50 mV/Div, 200 mV / Div,
500 mV/Div, 1 V/Div, 2 V/Div, 4 V/Div,
8 V/Div, 16 V/Div und 25 V/Div



Modus-Menü:
Einstellen des Modus für vorher selektierten Kanal. Mögliche Werte:
AC: Wechselanteil
DC: Gleich + Wechselanteil
GND: Aus

3.: Programm Code, Schnittstellen:

Zur Erstellung der Software wurde das GBDK verwendet, welches aus einem C Compiler, Assembler, Linker und Bibliotheken besteht. Er bringt Bibliotheken zum zur Benutzung im 8Bit Gameboy Code mit, z.B zum Zugriff auf Hardware. Die Compilier-Anweisung in unserem Makefile ist folgende:

..\..\gbdk\bin\lcc -Wl-j -Wl-ya1 -Wl-yt8 -Wl-z -o SLOT\gbdso.gb gbdso.c

Das Programm lcc dient als Frontend für den Compiler, Assembler und Linker. Erklärung der Parameter:

- Wl-j : Symbol-File für den no$gmb Emulator erzeugen (Debugging)
- -Wl-ya1 : ext.RAM Bank wird eingeblendet. (wird an 0x0149 geschrieben)
- -Wl-yt8 : Code für für ROM + RAM (wird an 0x0147 geschrieben)
- -Wl-z : Linkeroption, File wird im Gameboy Format geschrieben
- -o SLOT\gbdso.gb gbdso.c :Output- und Input Files

auf diese Weise wird ein gbdso.gb File geschrieben welches im Emulator ausgeführt werden kann, oder mit entsprechender Hardware auf ein EEPROM geschrieben werden kann.

Hauptschleife des Programms (dso/dsotime.c, gekürzt):

```
[…] Includes […]

// Datenspeicher d. 2 Kanäle
// Arrays haben nur max. 127 Elemente
int pointsa1[70];
int pointsa2[70];
int pointsb1[70];
int pointsb2[70];

int pointsa1_old[70]; // alte Werte zwischenspeicher
int pointsa2_old[70];
int pointsb1_old[70];
int pointsb2_old[70];
```

```
// Speicher Menue
int menuitem;
int menustate;
int nextmenustate;

// Einstellungen:
unsigned timediv = 10; // ms/DIV, Wertebereich 1ms bis 100ms -> wird mit 10
multip.

unsigned cha_amp = 0; // 10 mV/Div ^= 1, 50 mV/Div, 200 mV / Div, 500mV/Div,
                      // 1 V/Div, 2 V/Div, 4 V/Div, 8 V/Div, 16 V/Div und
                         25 V/Div
unsigned chb_amp = 0;

// 10 [0-9] Werte a 4 Zeichen
char ch_amp_werte[40] = "10mV50mV200m500m1 V 2 V 4 V 8 V 16 V 25V ";

char cha_mode[3] = "DC "; // Mode, Möglichkeiten: AC(A)-DC(D)-GND(G)
char chb_mode[3] = "DC ";

void dsotime()
{
       koordinaten();
       dsomenu(&timediv, &cha_amp, &chb_amp, cha_mode, chb_mode);
       init_data(pointsa1, pointsa2, pointsb1, pointsb2, pointsa1_old,
                    pointsa2_old, pointsb1_old, pointsb2_old);

       menuitem = 1; // 1. Item
       menustate = 1; // 1. Item


       while(1){

              // alte punkte löschen
              // del_points(pointsa, pointsb);
              //Punkte werden jetzt direkt in print_data gelöscht

              // menü machen
              dsomenu(&timediv, &cha_amp, &chb_amp, cha_mode, chb_mode,
                       &menuitem,&menustate, &nextmenustate, ch_amp_werte);


              // Daten zwischenspeichern
              copy_data(pointsa1, pointsa2, pointsb1, pointsb2, pointsa1_old,
                           pointsa2_old, pointsb1_old, pointsb2_old);

              // Daten von der Schnittstelle lesen
              read_data(pointsa1, pointsa2, pointsb1, pointsb2);

              //  Daten aufbereiten
              transform_data(pointsa1, pointsa2, pointsb1, pointsb2,
                                  &cha_amp, &chb_amp, cha_mode, chb_mode);

              // Daten ausgeben
              print_data(pointsa1, pointsa2, pointsb1, pointsb2,
                     pointsa1_old, pointsa2_old, pointsb1_old, pointsb2_old);

       }
}
```

Erklärung:

Der Wertespeicher der Anzeige muss pro Kanal auf 2 Arrays (pointsa1 + pointsa2) aufgeteilt werden, da die maximale Arraygröße 127 beträgt (unsigned int). Die vom Benutzer gewählten, bzw Defaulteinstellungen befinden sich in den Variablen timediv, chx_amp, und chx_mode. Sie können in den Untermenüs direkt verändert werden. Die Verwendung von globalen Variablen ist geboten, da aufgrund der sehr beschränkten Stack-Größe des GB-Prozessors möglichst wenige Funktionsparameter verwendet werden sollten. Bei uns werden die Parameter hauptsächlich per Referenz übergeben.
Zur Funktion dsotime():
Die Hauptfunktion des Oszis wird direkt nach den Auswahlmenüs aufgerufen. Im ersten Schritt wird das Koordinatensystem gezeichnet, Das Benutzermenü am unteren Bildrand gezeichnet, und die Datenspeicher initialisiert.
Danach beginnt die Hauptschleife, die das Programm unaufhörlich durchläuft:

- Menübehandlung
- aktuelle Daten in Zwischenspeicher kopieren.
- neue Daten von der Schnittstelle lesend
- die Daten auf das Koordinatensystem transformieren
- Neue Daten zeichnen, und die alten Punkte löschen

Das Zwischenspeichern der Daten erfolgt aus Performancegründen, damit das Löschen und Neuzeichnen der Punkte quasi gleichzeitig erfolgen kann. So wird ein „flackern“ des Graphen vermindert.

Menüfunktionalität steckt in der Datei dso/dsomenu.c :

```
[...]
switch (joypad(0)){
            case (J_RIGHT):
                  *menuitem += 1;
                  [...]
                  break;
            case (J_LEFT):
                  *menuitem -= 1;
                  [...]
                  break;
            case (J_SELECT):
                  *menustate = *nextmenustate;
                  *menuitem = 1; // menuitem zurücksetzen
                  break;

            // zurück zum hauptmenü:
            case (J_START):
                  *menustate = 1;
                  *menuitem = 1; // menuitem zurücksetzen
                  break;
      }
```

Erklärung:

Hinter J_RIGHT, J_LEFT usw verbergen sich Makros zum abfragen de Bedienknöpfe.

- `menustate` ist eine id für das aktuell angezeigte Untermenu
- `menuitem` ist die id des aktuell hervorgehobenem Items

So habe ich versucht eine Struktur in die Menüführung zu bringen, und mehrfache Programmierung zu vermeiden.

Kommunikation mit dem Mikrocontroller zur Einstellung der Amplitude A/B, und Modus:

Zur Kommunikation wird die Adresse 0xA100U erwendet, an der der Gameboy ein externes RAM erwartet, und somit beschrieben werden kann. Da nur 8 Bit zur Verfügung stehen muß der Befehl kodiert werden.

Schematisch:

```
frei frei b5   b4   b3   b2   b1   b0  [Bits 7 – 0]
  |    |   |    |    |    |    |    |
  0    0   0    0    0    0    0    0   [gefüllt mit Nullen]
```

bits 0-3: Amplitude des Kanals mit Index 0 bis 9 (^= 10mV bis 25 V)
b4: Modus: 0:AC 1: DC
b5: Kanal: 0:A 1: B

Beispiel:

```
KanalB        -> 0   0   1   0   0   0   0   0 ^= $20
Modus DC      -> 0   0   0   1   0   0   0   0 ^= $10
Amp: 500mV    -> 0   0   0   0   0   0   1   1 ^= $03
```

Der in 0xA100U geschriebene Wert wäre hier: $33