Gigatron Micro

V1.50 (c) 2018-2019 Jörg Wolfram

1 Rechtliches

Die Programme unterliegen der BSD 2-Clause Lizenz.

Die Veröffentlichung dieses Projekts erfolgt in der Hoffnung, daß es Ihnen von Nutzen sein wird, aber OHNE IRGENDEINE GARANTIE, auch ohne die implizite Garantie der MARKTREIFE oder der VERWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK.

Das Projekt wurde unter Linux entwickelt, eine Kompatibilität zu anderen Betriebsystemen ist nicht garantiert. Die bereitgestellten Binärfiles sind aber betriebsystem-unabhängig.

Alle im Text genannten Marken sind Eigentum des entsprechenden Inhabers.

2 Wozu?

Das Projekt emuliert einen von Marcel van Kervinck und Walter Belgers entwickelten Einplatinen-Computer auf TTL Basis
(**https://gigatron.io**).
Hauptgrund war aber die Herausforderung, auf einem modernen 32-Bit Controller zyklengenau programmieren zu können und mich mit ARM-Assemblercode zu beschäftigen.

3 Was wird emuliert?

Es wird ein Gigatron mit 64K RAM emuliert, die „externe" Hardware ist praktisch identisch zum Original (bis auf die niedrigeren Spannungspegel). Um das erreichen zu können, wird der Controller auf 225 MHz übertaktet und hat damit 36 Takte zur Verfügung, einen Gigatron-Befehl auszuführen.
Leider ist die Emulation nicht 100%-ig, da ich es nicht geschafft habe, die steigende HSYNC-Flanke für die I/O auszuwerten. Das ließ sich aber durch kleine Patches am ROM-Image und unbenutzte Befehle ausgleichen. Anstelle von 0x18 (ld $dd,out) verwende ich 0x1C für die steigende HSYNC-Flanke und anstelle von 0x19 (ld [$dd],out) einen äquivalenten Befehl unter 0x1D. Für die bestehenden ROM-Images (v1-v3) übernimmt das Tool für die Konvertierung in ASM-Quellcode auch gleich das Patchen mit. Dabei wird auch der Instruction-Code in das High-Byte geschrieben.

4 Wie wird emuliert?

Das Projekt ist vollständig in Assembler geschrieben und nutzt selbst kein RAM (auch keinen Stack). Alle Informationen werden in Registern gehalten. Das ausgewählte ROM-Image wird beim Start in das RAM des Controllers kopiert und das CCM RAM dient als emuliertes RAM. Der Emulator selbst läuft aus dem Flash (mit eingeschaltetem Prefetch). Dafür gibt es 256 kleine „Programm-Inseln", welche jeweils einen Befehl ausführt und dann zur Programminsel für den nächsten Befehl springt. Da der Instruction-Code im High-Byte liegt, lässt sich über das Ausmaskieren der unteren 8 Bits und Addition der Startadresse leicht die Adresse des nächsten Befehls ermitteln. Als Video-Port dienen die unteren 8 Bits von Port A, Bit A8 ist SER_DATA. Über Port C0-C7 werden LEDs und Audio angesteuert. Zusätzlich gibt es noch zwei LEDs an PORT B5 und B7, diese haben mir hauptsächlich während der Entwicklung gedient und sind eigentlich nicht zwingend notwendig.

5 Wahl des ROM Images

Standardmäßig wird das v3 ROM geladen. Wir während des Power-Up (bis beide Status LED leuchten) die Taste A auf dem angeschlossenen Game-Controller gehalten, wird stattdessen das v1 ROM geladen, entsprechend bei Taste B das v2 ROM.

6 Selbst übersetzen

Im **bin**-Verzeichnis gibt es ein gtmicro.s39 File, welches direkt auf den Controller geflasht werden kann. Um den Quellcode zu übersetzen gibt es im **source**-Verzeichnis ein Makefile, in dem noch der Pfad zur auf dem System befindlichen ARM-NONE-EABI Toolchain eingetragen werden muss. Der Programmierbefehl (make prog) ist dabei an meinen UPROG2 Programmer angepasst.

7 Changelog

7.1.2019 Version 1.50

- erste öffentliche Version