Ein Wort zuvor

ST baut zwar in alle seine STM32 Controller einen residenten Bootlader ein, doch um dessen Unterstützung durch eine passende PC-Software ist es schlecht bestellt. Deren „STMFlashLoaderDemo“ ***ist*** lausig. Punkt. Deshalb habe ich mir mal selbst so etwas zurecht geschrieben, was mittlerweile benutzbar ist. Besonders schön und elegant ist es nicht und diverse Einzelheiten gehen aus prinzipiellen Gründen nicht. Dazu zählt vor allem eine anfangs von mir ins Auge gefaßte automatische Chip-Bestimmung. Ich hatte die Map-Dateien des STMFlashLoaders durchgesehen, um zu einer Chipauswahl für mein Programm zu kommen und mußte dabei feststellen, daß offenbar die gleiche PID an verschiedene Chips vergeben wurde. Es möge also ein jeder Interessent die Datei „targets.cfi“ nach seinem Gusto bearbeiten, um seinen Lieblings-Chip einzutragen und nicht im Gewusel von ST den Überblick zu verlieren.

Was ist das Ganze überhaupt?

Das vorliegende Programm dient zum Programmieren von STM32Fxxx Controllern unter Verwendung von deren fest eingebautem Bootlader über eine serielle Schnittstelle. Es ist mit Delphi 6 geschrieben und läuft demzufolge unter Windows auf dem PC. Die Quellen sind mit dabei, so daß ein interessierter Linux-Benutzer das Ganze auch mittels Freepascal/Lazarus für sein BS herrichten kann.

Was braucht man und wie funktioniert es?

Zunächst braucht man eine serielle Schnittstelle, die mit TTL-Pegeln bzw. 3.3 Volt Pegeln auf der seriellen Seite arbeitet. Im Zweifelsfalle nachmessen! Wir können hier keine echten V.24 Pegel gebrauchen.

Ich benutze einen simplen USB-Adapter, auf dem ein FT232BL von FTDI verbaut ist. Dessen TxD und RxD Signale gehen beim STM32 an dessen Default-Pins für USART1_RXD bzw. _TXD (immer TxD von einer Seite an RxD der anderen Seite).

Dazu werden aber auch DTR und RTS des FTDI-Chips zum Reset des STM32 und zum Auswählen des Bootlader-Starts benutzt.

Natürlich kann man auch andere Chips nehmen, solange man dort an DTR und RTS herankommt. Viele billige China-Adapter mit einem Prolific-Chip haben diese Signale leider ***nicht*** herausgeführt.

Mit einem dieser Signale wird der Reset-Eingang des STM32 verbunden und mit dem anderen wird der Umschalteingang (BOOT_0) verbunden – welcher wohin führt, kann man im Programm einstellen.

Manche Chips haben dazu noch einen weiteren Boot-Pin (BOOT_1). In diesen Fällen muß man im zugehörigen Referenzmanual nachlesen, welche Kombination aus BOOT_0 und BOOT_1 zu welchem Startverhalten führt. Zum Programmieren muß der Controller nach dem Reset in den Bootlader gehen und nicht zum Flash-Speicher.

Wichtig: Man kann am STM32 die Boot-Pins auch per Jumper o.ä. festlegen, aber den Zugang zum Reset-Pin braucht man in jedem Falle, weil der chipinterne Bootlader bei einigen Löschvorgängen den Chip per Software-Reset zurücksetzt. Das klappt nach meiner Erfahrung nur manchmal richtig und deshalb ist es einfach sauberer und zuverlässiger, vom PC-Programm aus den echten Hardware-Reset zu betätigen.

Natürlich muß man sein Board mit dem Chip drauf zum Programmieren mit Strom versorgen – das sei nur der Vollständigkeit halber erwähnt.

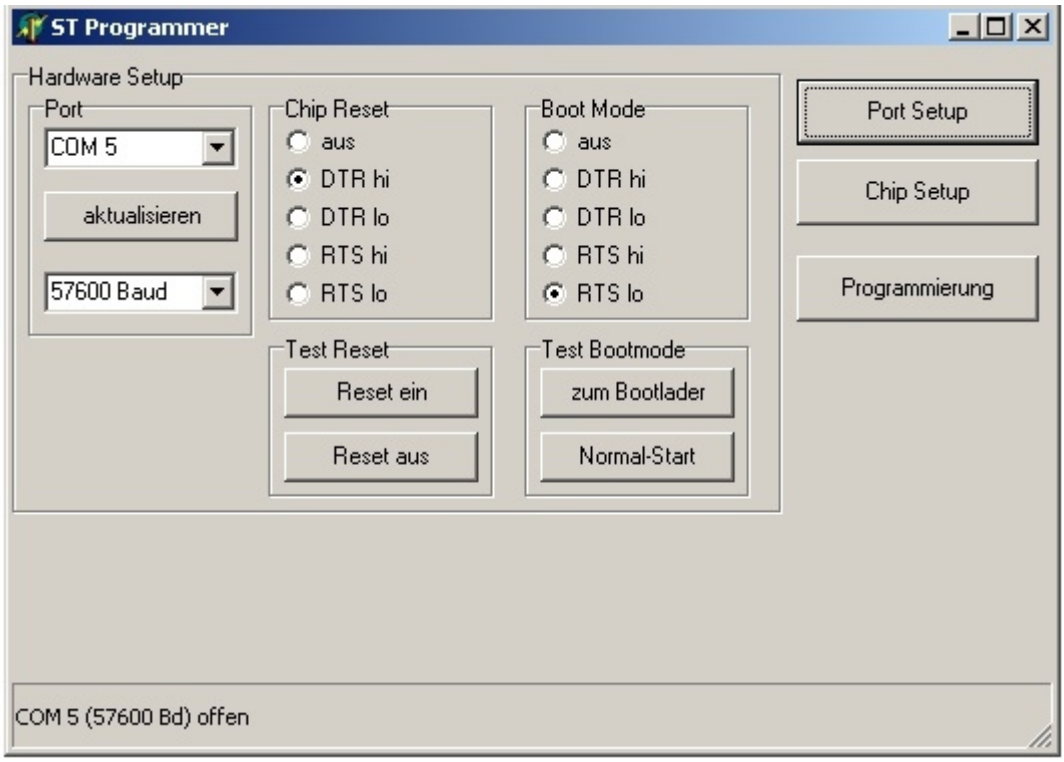
Zum Programmieren braucht man ein Hexfile – und zwar eines im Intel-Hex-Format. Das muß die Firmware enthalten, und zwar auf die tatsächlichen Ausführungsadressen gelinkt. Klingt kompliziert, ist es aber gar nicht. Normale Cortexe greifen ab Reset auf Adressen ab 0 (null) zu. Also muß dort wenigstens die Vektortabelle stehen und der Code dann dort hintendran. Aber der programmierbare Flash steht bei allen mir bekannten STM32 ab Adresse 0800'0000h an. Deswegen portiert mein Programm allen Code im Hexfile auf den Bereich ab Flash-Anfang. Später beim normalen Programmstart wird dieser Bereich dann auf Adresse 0 gespiegelt und die gewöhnliche Abarbeitung kann dann ab dort stattfinden.

Eigentlich habe ich geplant, auch noch Motorola-Hexfiles und Binärfiles lesbar zu machen, aber das kommt irgendwann mal…

Die Programmbeschreibung

Zunächst der Aufruf: normalerweise ohne Parameter. Wird ein Parameter angegeben, dann erwartet das Programm entweder eine Konfigurationsdatei (*.cfp) oder eine Hexdatei (*.hex) – beides mit vollem Pfad.

Nun das Programm, wenn man den Port Setup Knopf gedrückt hat:

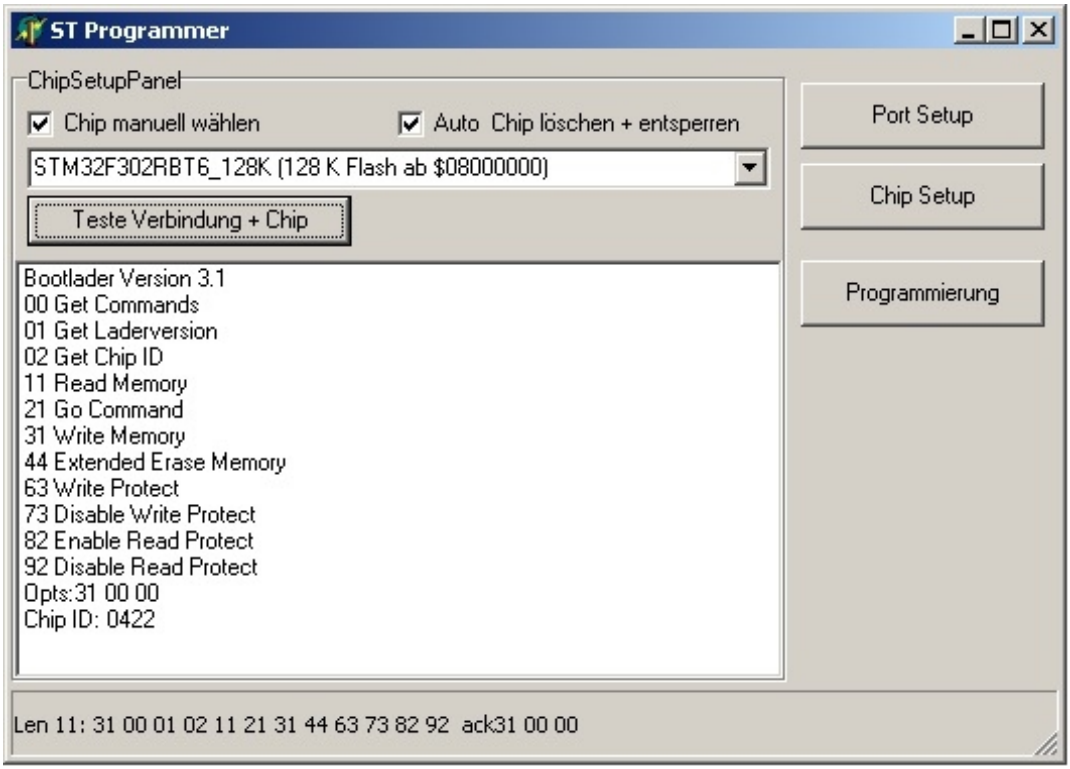


Hier sucht man sich den passenden COM-Port aus, unter dem man seinen Programmieradapter erreicht. Beim Programmstart scannt das Programm alle Ports von COM1 bis COM99 und trägt nur die Ports in die Liste ein, die es im System momentan auch gibt. Steckt man also seinen Adapter erst nach dem Programmstart ein, kann man die Liste mit dem darunter liegenden Knopf aktualisieren lassen.

Bei den Baudraten werden nur die üblichen Raten angezeigt.

Die Signale für den Chip-Reset und für den Boot-Modus können rechts davon ausgewählt werden. Die darunterliegenden Knöpfe dienen zum Testen mittels Multimeter auf der Leiterplatte, wo sich der zu programmierende Chip befindet.

Hier sieht man das Programm, wenn der Chip Setup Knopf gedrückt wurde:



Die beiden Checkboxen oben sind derzeit funktionslos. Darunter ist die Auswahl-Liste für den gewünschten STM32-Typ. Diese enthält das Zeugs, was ich von der „STMFlashLoaderDemo“ aus deren Setup-Dateien entnommen habe. Wenn der gewünschte Typ nicht dabei ist, dann muß man per Texteditor die Datei „targets.cfi“ bearbeiten und seinen STM32 dort eintragen. Jede Zeile enthält einen Chip nach dem Muster:

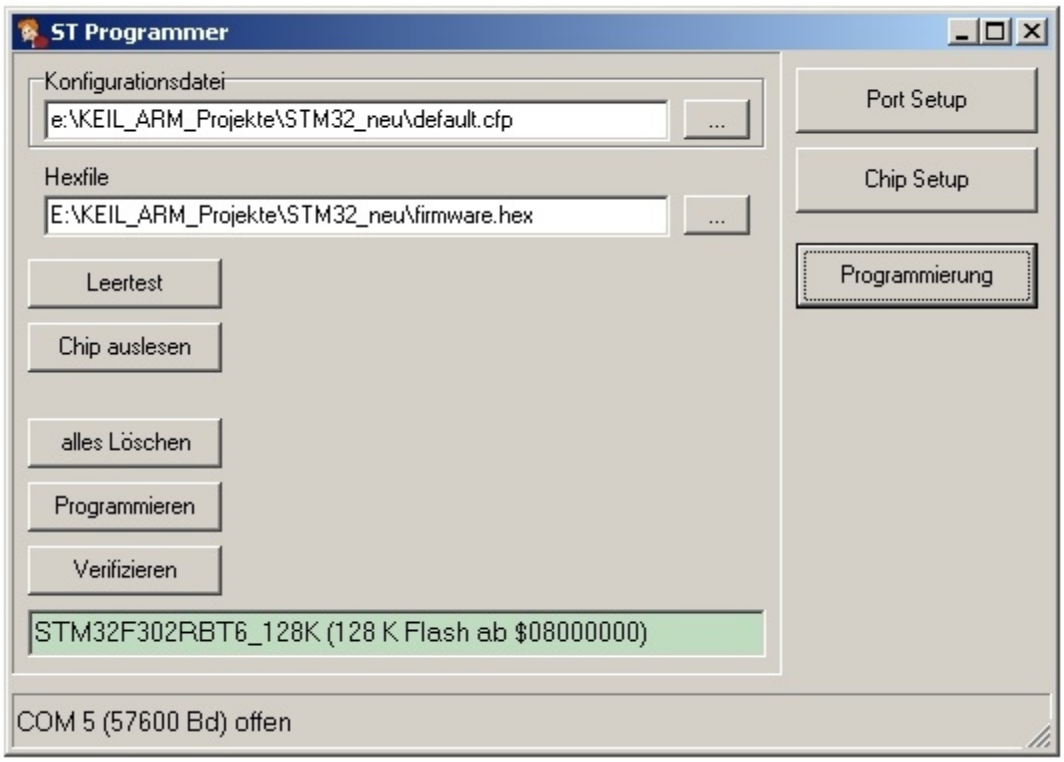
Chipname; PID; Flash-Anfang; Sektorgröße; Flash_Ende; Optionadresse

wobei alles außer dem Chipnamen in HEX geschrieben ist und alle Spalten durch Semikolon getrennt sind. Die Sektorgröße und Optionadresse verwende ich derzeit nicht. Es gibt also bislang keine Auslese-Sperre usw.

Mit dem unter dieser Liste liegenden Test-Knopf kann man testen, ob die Verbindung vom PC zum STM32 funktioniert, ob der Bootlader richtig startet und was er für Kommandos verträgt. Wenn man im Log-Fenster das sieht, was in obigem Bild zu sehen ist, dann ist die Verbindung OK.

Ich habe all diese Testmöglichkeiten im Port-Setup und im Chip-Setup mit Bedacht vorgesehen, damit man sein Programmiergeschirre möglichst problemlos in Gang bekommt - das grottenschlechte Beispiel des „STMFlashLoaders“ vor Augen. Der kaut bloß stumm auf einem etwaigen Problem herum – und mit der finalen Meldung „es geht nicht“ kann man fast gar nix anfangen.

Kommen wir nun zur Haupt-Ansicht des Programms:



Hier sieht man, welche Konfigurationsdatei aktiv ist, welches Hexfile zu programmieren ist und welcher konkrete Chip ausgewählt ist.

Ein Wort zur Codegröße: Ich habe Pufferspeicher für maximal $200000 Bytes, also 2 MB vorgesehen. Ich denke, mit diesen maximal 2 MB Code wird man hier erst einmal auskommen.

Eine durchlaufende Automatik habe ich vorläufig noch nicht vorgesehen, also muß man es in Einzelschritten tun: alles löschen, programmieren, verifizieren. Ein „mach Alles“-Knopf kommt später mal auf die rechte Seite.

Eine Sonderstellung nimmt das Chip-Auslesen ein. Es wird zum Programmieren nicht benötigt und es liest immer den gesamten Flash des Chips aus und speichert dessen Inhalt in einer Binärdatei.

Vorsicht: Das Auslesen und der Leertest können recht lange dauern, da alle Daten in nur 256 Byte kleinen Stückchen ausgelesen werden können und man in vielen Fällen mit der Baudrate nicht zu hoch einsteigen darf.

Viel Spaß beim Basteln wünscht

W.S.