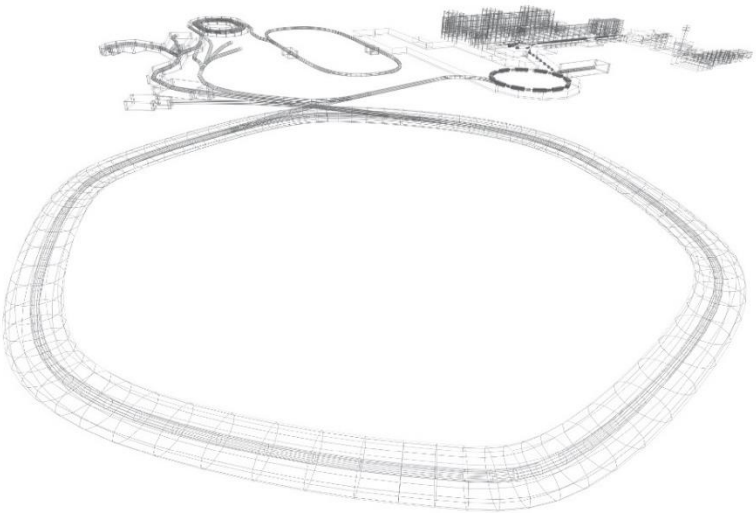
Elektrische
Energie
Technik

GSI Darmstadt



MFU via JTAG programmieren und ACU Remoteupdate durchführen

Version vom: Donnerstag, 9. Juli 2020, 11:51:00

Aktuelle Versionen immer unter:

https://wiki.gsi.de/foswiki/bin/view/EPS/ACUDocuments

Inhaltsverzeichnis

1. Änderungsliste ... 1
2. Abkürzungen ... 2
3. Einleitung ... 3
   3.1. FPGA FW Image ... 3
   3.2. NiosII Software ... 3
4. Eine neue MFU programmieren ... 5
   4.1. 1. Schritt - TFT Displaymodul der MFU ... 5
   4.2. 2. – 4. Schritt - automatisch mit der MultiFunctionUnit_Boot.flash ... 6
      4.2.1. Die MultiFunctionUnit_Boot.flash Datei ... 6
      4.2.2. MultiFunctionUnit_Boot.flash nutzen ... 6
         4.2.2.1. Was wird dazu benötigt ... 6
         4.2.2.2. Vorbereitung ... 6
         4.2.2.3. Der eigentliche Bootvorgang ... 7
      4.2.3. Weitere Programmierschritte via USB Speicher ... 8
         4.2.3.1. Bedingungen für die Verwendung des USB Speicher ... 8
         4.2.3.2. Die update.txt Datei ... 8
         4.2.3.3. Was tun beim Abbruch ... 9
         4.2.3.4. Die Einträge und ihre Bedeutungen ... 9
   4.3. 2. Schritt manuell - FPGA FW programmieren ... 11
   4.4. 3. Schritt manuell - NiosII SW programmieren ... 11
      4.4.1. MFU_Flash_cfi_flash_ABC_FAC_APP.flash Datei programmieren ... 11
         4.4.1.1. Was wird dazu benötigt ... 11
         4.4.1.2. Vorbereitung ... 11
         4.4.1.3. Der eigentliche Programmiervorgang ... 11
         4.4.1.4. Ein paar warme Wort zu „prog_mfu.sh“ ... 11
   4.5. 4. Schritt manuell VNC2 SW programmieren ... 12
   4.6. 5. Schritt manuell FTDI Chip Seriennummer programmieren ... 12
5. Die Programmierdateien erstellen ... 14
   5.1. Erstellen der MultiFunctiononUnit_Boot.flash für den EPCS ... 14
      5.1.1. Was wird dazu benötigt ... 14
      5.1.2. Vorbereitung ... 14
      5.1.3. Der eigentliche Konvertierungsvorgang ... 14
      5.1.4. Ein paar warme Worte zu „gen_mfu_boot.sh“ ... 15
   5.2. Erstellen der NiosII SW MFU_Flash_cfi_flash_xxx.flash Dateien für das Parallel-Flash ... 16
      5.2.1. Was wird dazu benötigt ... 16
      5.2.2. Vorbereitung ... 16
   5.3. Der eigentliche Konvertierungsvorgang ... 16
6. Dinge die der Entwickler wissen MUSS ! ! ! ... 18

1. Änderungsliste

Datum	Name	Kommentar
01.11.2010	D. Schupp	Dokument erstellt
10.01.2010	D. Schupp	Kapitel: Erstellen der Programmierdateien hinzu gefügt
21.01.2011	D. Schupp	Dokument ergänzt um die Beschreibung des ACU RemoteUpdate. Dokumentenname dahingehend erweitert. Dadurch wird das Dokument ACU_Update obsolet.
06.03.2013	D. Schupp	Dokument für QuartusII V11.1 überarbeitet
19.08.2013	D. Schupp	Kapitel 2 neu
23.07.2014	D. Schupp	Kapitel 4.2.3 „Weitere Programmierschritte via USB “ überarbeitet
16.09.2015	D. Schupp	Aktualisierung von Kapitel: 4.2.3.2 „Die update.txt Datei“
04.11.2016	D. Schupp	JTAG
12.10.2018	D. Schupp	Vollständig überarbeitet
19.02.2020	D. Schupp	Kommentarzeilen in der update.txt sind möglich
05.03.2020	D. Schupp	Details zum Kommando *reconf
09.07.2020	D. Schupp	Info über max. Länge von Kommentarzeilen in update.txt

2. Abkürzungen

ACU ***A**daptive **C**ontrol **U**nit* - alle zur Steuer- und Kontrolleinheit eines Netzgerätes gehörenden Komponenten

ABC ***A**dvanced **B**oot **C**opier* - wählt die richtige NIOSII SW für die jeweils geladene FPGA FW aus

MFU ***M**ulti **F**unction **U**nit* - zentrale Grundeinheit eines ACU-Systems

FPGA ***F**ield **P**rogrammable **G**ate **A**rray* - ein programmierbarer Logikbaustein der die Funktionalitäten eines ACU Moduls beinhaltet

VNC2 ***Vinc**ulumII* - ein Mikroprozessor der eine USB Schnittstelle zur Verfügung stellt

NIOSII Mikroprozessor in der MFU der die Benutzerschnittstelle zur Verfügung stellt

FW ***F**irm**w**are* - die notwendige Programmierung des FPGA. Es wird unterschieden zwischen Application und Factory Image. Das Application Image ist i.d.R. durch den Endanwender ersetzbar. Das Factory Image hingegen nur durch das Support-Team.

SW ***S**oft**w**are* - Betriebssoftware der NIOSII/VNC2 Mikroprozessoren innerhalb der MFU. Auch hier wird unterschieden zwischen Application und Factory Image. Das Application Image ist i.d.R. durch den Endanwender ersetzbar. Das Factory Image hingegen nur durch das Support-Team.

EPCS serieller Flash-Speicher - beinhaltet die FPGA Firmware (FW)

.RBF ***R**aw **B**inary **F**ile* - enthält eine FPGA FW Konfiguration in Binärdarstellung. Eine RBF Datei wird zum Programmieren eines FPGA via Remote oder über einen USB Flash Speicher benutzt und enthält i.d.R. nur das FPGA FW Application Image

.JIC ***J**TAG **I**ndirect **C**onfiguration*, enthält eine oder mehrere FPGA FW Konfigurationen. Eine JIC Datei wird zum Programmieren eines FPGA mittels USB-Blaster benutzt und enthält i.d.R. das FPGA FW Application und das Factory Image.

.FLASH Bei der .flash-Datei handelt es sich um eine Programmierdatei im Motorola S19 Format. Diese beinhaltet i.d.R. die NiosII Application SW, die NiosII Factory SW und den notwendigen Boot Kopierer (ABC)

.S19 Ist ebenfalls eine Programmierdatei im Motorola S19 Format, beinhaltet aber nur die NiosII Application SW.

3. Einleitung

3.1. *FPGA FW Image*

Jedes FPGA wird mit 2 unterschiedlichen FPGA FW-Konfigurationen programmiert, dem Factory-Image und dem Application-Image. Dazu wird der Gesamtspeicher des EPCS welcher die FPGA Konfiguration beinhaltet zweigeteilt. In der ersten Hälfte befindet sich das Factory-Image, in der zweiten das Application-Image.

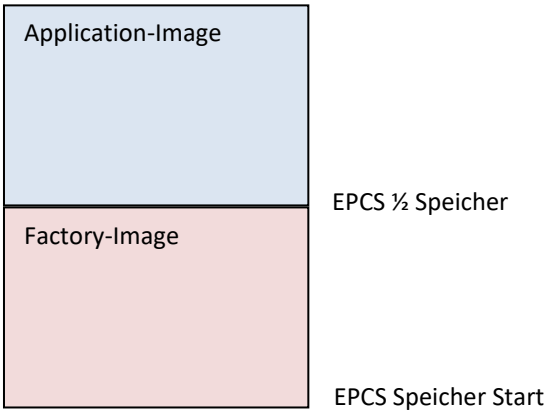


Abbildung 3.1: EPCS Speichernutzung für FPGA FW Images

Das FPGA startet immer im Factory-Image und prüft anschließend ob ein Application-Image vorhanden ist. Ist dies der Fall startet das FPGA erneut im Application-Image.

Der Endbenutzer aktualisiert i.d.R. nur das Application-Image. Schlägt dies fehl, startet das FPGA im Factory-Image und ermöglicht den erneuten Versuch das Application-Image zu programmieren.

Es gibt verschiedene Wege das Application-Image zu ersetzte.

- Lokal vor Ort via USB-Blaster und QuartusII mit einer `.jic` (JTAG Indirect Configuration) Datei. Diese enthält zusätzlich immer das Factory Image. Es werden also immer beide FPGA Konfigurationen ersetzt.
- Lokal vor Ort via USB Flash Speicher mit einer `.rbf` (Raw Binary File) Datei
- Via Remote ebenfalls mit einer `.rbf` (Raw Binary File) Datei

3.2. *NiosII Software*

Für jedes FPGA Image (jede FPGA FW-Konfiguration) existiert eine eigene NiosII Software, da die NiosII-Software im Factory-Image und Application-Image nicht zwangsläufig identisch sein muss. Außerdem existiert eine dritte Software, der sogenannte Advanced Boot Copier (ABC). Dieser sorgt dafür, dass die jeweils richtige, zum FPGA FW Image gehörige Software vom NiosII ausgeführt wird.

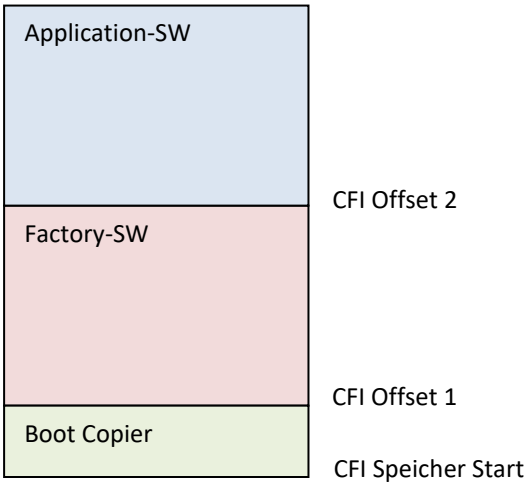


Abbildung 3.2: Speichernutzung für NiosII Software

Nachdem das jeweilige FPGA FW Image geladen wurde, startet der NiosII den Advanved Boot Copier, welcher abhängig vom geladenen FW Image entweder die Factory- oder die Application-SW für den NiosII zur Abarbeitung zur Verfügung stellt.

Der Endbenutzer aktualisiert i.d.R. nur die NIOSII Application-SW. Schlägt dies fehl, startet das FPGA mit der Factory-Image FW und ermöglicht den erneuten Versuch die NiosII Application-SW zu programmieren.

Es gibt verschiedene Wege die Application-SW zu ersetzte.

- Lokal vor Ort via USB-Blaster und der NiosII SBT von QuartusII mit einer `.flash` Datei. Diese enthält zusätzlich immer die Factory SW. Es werden also immer beide Softwaren ersetzt.
- Lokal vor Ort via USB Flash Drive mit einer `.s19` Datei.
- Via Remote ebenfalls mit einer `.s19` Datei.

4. Eine neue MFU programmieren

Eine neue und bisher vollständig nicht programmierte MFU wird in den folgenden Schritten mit Firm- und Software programmiert:

1. TFT Displaymodul der MFU programmieren
2. MFU FPGA FW programmieren
3. MFU NiosII SW programmieren
4. MFU VNC2 SW Programmieren
5. FDTI Chip Seriennummer programmieren

Schritt 1 ist in jedem Fall als Einzelschritt notwendig.

Die Schritte 2-4 lassen sich halbautomatisch im Ganzen oder auch einzeln, manuell durchführen.

Schritt 5 ist in jedem Fall als Einzelschritt notwendig.

4.1. 1. Schritt - TFT Displaymodul der MFU

Dazu

1. USB Blaster an JTAG anschließen,
2. QuartusII Programmer starten und
3. die Datei `MFU_TFTFrontPanel_FandA.jic` programmieren.

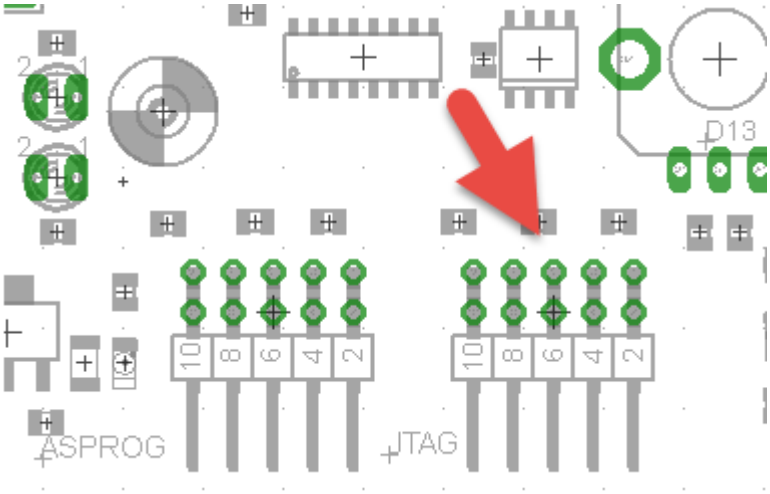


Abbildung 4.1: JTAG Anschluss auf der MFU TFT Front Platine

4. Im Anschluss das TFT Displaymodul der MFU kurz von der Versorgungsspannung trennen damit dieses neu startet.

4.2. 2. – 4. Schritt - automatisch mit der MultiFunctionUnit_Boot.flash

4.2.1. Die MultiFunctionUnit_Boot.flash Datei

Bei einer neuen MFU sind alle Speicher leer. Es muss zunächst eine FPGA FW programmiert werden um Zugriff auf das CFI–Flash (ein Flash mit Common Flash Interface) zu erhalten um die zugehörige NiosII SW zu programmieren. Mit Hilfe dieser NiosII SW ist es schließlich möglich den VNC2 (VinculumII USB Host Controller) zu programmieren mit welchem USB Speichermedien verwendet werden können.

Dieser Programmiervorgang kann automatisiert werden. Dazu muss als erstes das FPGA Boot FW Image in das EPCS programmiert werden.

Diese Boot FW beinhaltet zusätzlich eine kleine NiosII Boot SW und die notwendige Programmierdatei für den VNC2.

Nach dem Programmieren wird das FPGA mit der Boot FW gestartet. Danach startet der darin hineinkompilierte NiosII die ebenfalls im Boot FW befindliche Boot SW. Dadurch erhält das FPGA Zugriff auf den EPCS und ist in der Lage ab Speicherbereich 0x150000 die VNC2 SW zu lesen und den VNC2 zu programmieren.

War all das erfolgreich kann über einen am VNC2 angeschlossenen USB Speicher die weitere Programmierung der MFU durchgeführt werden.

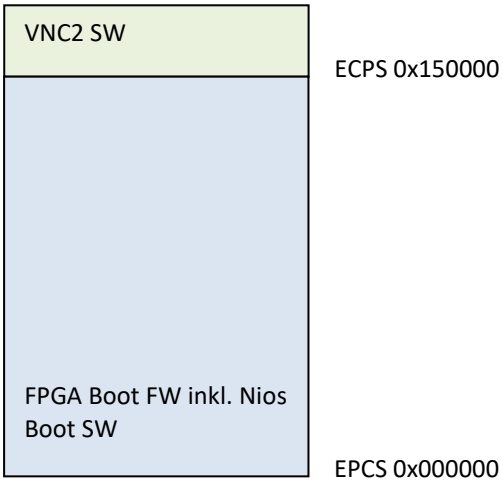


Abbildung 4.2: Speichernutzung des EPCS bei MultiFunctionUnit_Boot.flash

Diese erste Programmierdatei ist die Datei `MultiFunctionUnit_Boot.flash`.

Sofern `MultiFunctionUnit_Boot.flash` erst erstellt werden muss, siehe Kapitel 5.1: „Erstellen der MultiFunctiononUnit_Boot.flash für den EPCS“ auf Seite14.

4.2.2. MultiFunctionUnit_Boot.flash nutzen

4.2.2.1. Was wird dazu benötigt

1. Das NIOSII Terminal.
   Als Teil der NiosII SoftwareBuild Tools installiert, die wiederrum Teil von OuartusII sind.

4.2.2.2. Vorbereitung

1. Ins Installationsverzeichnis der NiosII EDS wechseln (`.../nios2eds/`).
2. Sofern vorhanden dort die Datei „`user.bashrc`“ entfernen oder umbenennen z.B. in „`_user.bashrc`“. Dabei handelt es sich um eine Skriptdatei die automatisch nach dem Start des NiosII Terminals ausgeführt wird.

3. Im Ordner mit Namen „bootfiles“ (.../nios2eds/bootfiles) werden die zuvor erstellte MultiFunctionUnit_Boot.flash und MultiFunctionUnit_Boot.sof Dateien gespeichert. Sofern MultiFunctionUnit_Boot.flash erst erstellt werden muss, siehe Kapitel 5.1: „Erstellen der MultiFunctiononUnit_Boot.flash für den EPCS“ auf Seite14.
4. Das Skript „boot_mfu.sh“ kommt ins Verzeichnis „.../nios2eds“.
5. Im Skript „boot_mfu.sh“ ist der Dateiname der MultiFunctionUnit_Boot.flash ggf. anzupassen.

4.2.2.3. Der eigentliche Bootvorgang

1. “Nios II Command Shell.bat” starten, dadurch öffnet sich das NiosII Terminal
2. ./boot_prog.sh im Terminal eingeben und ENTER drücken
3. Die Boot-Programmierung sollte starten und führt folgende Schritte durch:
   1. Es wird die FPGA MultiFunctionUnit_Boot.sof Datei in das FPGA programmiert.
   2. Das FPGA startet damit. Dadurch ist ein Zugriff auf das EPCS via JTAG und FPGA möglich.
   3. Es wird die MultiFunctionUnit_Boot.flash Datei ins EPCS programmiert.
   4. Das FPGA startet damit.

4.2.2.3.2. Nach Abschluss der MultiFunctionUnit_Boot.flash Programmierung

Wurde MultiFunctionUnit_Boot.flash erfolgreich ins EPCS der MFU programmiert sollte der NiosII automatisch oder durch kurzes entfernen der Betriebsspannung neu starten. Ist der VNC2 noch nicht programmiert wird dies als erstes durchgeführt. Sofern ein legitimierter USB Speicher mit den weiteren Programmierdateien zuvor im USB1 Anschluss steckt wird im Anschluss darüber die weitere Programmierung der MFU vorgenommen.

Andernfalls wird ein Fehlercode angezeigt, d.h. USB Speicher einstecken, die MFU neu starten und dann sollte die automatische Programmierung starten.

4.2.3. *Weitere Programmierschritte via USB Speicher*

Auf dem USB Speicher wird dazu nach der Datei „`update.txt`“ gesucht und diese wird abgearbeitet.

Die Datei „`update.txt`“ enthält dabei Informationen über die weiteren Schritte der MFU Programmierung. Um ein neues System zu programmieren oder ein bestehendes System zu aktualisieren ist es nötig die auf dem USB Speicher befindliche Datei „`update.txt`“ zu editieren und die entsprechend benötigten Programmierdateien ebenfalls darauf zu speichern.

Alle diese Dateien müssen im Rootverzeichnis zu finden sein.

Zu beachten ist auch, dass der USB Speicher nicht zu voll ist, damit die Dateisuche nicht zu lange dauert.

4.2.3.1. *Bedingungen für die Verwendung des USB Speicher*

Updatevorgänge über den USB Speicher sind nur möglich, sofern dieser dazu legitimiert ist. Diese Legitimierung führt die Netzgerätegruppe EPS der GSI durch. Unterstützt ein USB Speicher diese Option nicht, wird dies auf dem TFT durch:

```
The USB Device does not meet all the requirements. Please check USB
flash drive.
```

angezeigt.

Außerdem werden Updates nur durchgeführt, wenn das Netzgerät nicht in Betrieb ist, also das ACU System nicht eingeschaltet ist (Regler ist nicht freigegeben). Andernfalls erscheint im Display:

```
Unit is switched on. Please switch off go back and try again.
```

4.2.3.2. *Die update.txt Datei*

In der Datei „`update.txt`“ ist die sequentielle Reihenfolge der zu aktualisierenden Bauteile der MFU zu finden.

Beispiele:

1. Es soll die Application Software des NiosII auf der MFU aktualisiert werden:

```
0 0 mfuflash.s19
```

2. Es soll die Application FW des FPGA der MFU und die Application FW des FPGA des Modul 1 an USI 1 aktualisiert werden (z.B. ADC) aktualisiert werden

```
0 0 mfu_fpga.rbf
1 1 adc_fpga.rbf
```

usw.

Die einzelnen Einträge müssen in der richtigen Reihenfolge untereinander, jeweils durch Return getrennt aufgelistet sein. Nach dem letzten Eintrag wird kein Return benötigt.

Die Einträge und ihre Bedeutungen:

`usi_number  module_number  filename.file_extension`

`usi_number`	USI Nummer 1… USI Max (hexadezimal)
`module_number`	Modul Nummer 1… Modul MAX (hexadezimal)
`filename.file_extension`	Name der auszuführenden Datei nach DOS 8.3 Konvention

Sind `usi_number` und `module_number` 0 (Null), wird die MFU aktualisiert.

Zwischen `usi_number` und `module_number` und `filename.file_extension` muss sich jeweils 1 (ein) Leerzeichen befinden!

Der Name der auszuführenden Datei ist frei wählbar, muss aber die alte DOS 8.3 Formatierung einhalten. Die Dateierweiterung gibt an um welche Art von Datei es sich handelt. Dateien die ein FPGA FW

Application Image aktualisieren enden i.d.R. auf `.rbf`. Dateien die ein Application Image der Nios SW aktualisieren auf `.s19`.

„`update.txt`“ wird zeilenweise gelesen und mit den Einträgen einer temporär erzeugten Datei „`update.tmp`“ verglichen. Bevor das Update eines Moduls durchgeführt wird, wird der zugehörige Eintrag in die Datei „`update.tmp`“ kopiert und dann das Update durchgeführt. Sind alle Updates beendet wird „`update.tmp`“ wieder gelöscht.

Ein evtl. beim Update auftretender Fehler der zum Abbruch führt, wird in der „`update.err`“ vermerkt:

```
Wed 16.09.15 10:55:12 update process failed with error code (NIOS): 065 (VNC2): FI
```

Mit Datum, Uhrzeit, Fehlercode der Funktion (Nios) und Fehlermeldung des USB Host Controllers (VNC2).

4.2.3.3. Was tun beim Abbruch

Ist ein Update fehlgeschlagen, kann der „`update.tmp`“ genau entnommen werden, bei welcher Aktualisierung der Fehler auftrat und ggf. ein Fehlercode dazu in der „`update.err`“. Um das Update fortzusetzen, muss aus der „`update.tmp`“ der letzte Eintrag entfernt werden. Wird das Update dann erneut gestartet, setzt das Update am letzten Punkt vor dem Auftreten des Fehlers fort.

update.txt	update.tmp
0 0 mfu_fpga.rbf	0 0 mfu_fpga.rbf
0 0 mfuflash.s19	0 0 mfuflash.s19
1 1 icm_fpga.rbf	1 1 icm_fpga.rbf
2 1 adc_fpga.rbf	

Abbildung 4.3: Beispiel einer „update.txt“ zur „update.tmp“ nach erfolgtem Fehler

Wie in Abbildung 4.3 zu sehen ist, trat beim Update von „`icm_fpga.rbf`“ ein Fehler auf.

Um die Durchführung von „`icm_fpga.rbf`“ nun neu zu starten, muss aus der „`update.tmp`“ die letzte Zeile entfernt werden (gelb markiert), inkl. des `<CR>` und `<LF>` der Zeile „`mfuflash.s19`“, d.h. der Cursor muss hinter der ‚9’ des „`mfuflash.s19`“ Eintrags platziert sein.

4.2.3.4. Die Einträge und ihre Bedeutungen

4.2.3.4.1. Dateinamen

Der VNC2 USB-Master kann nur Dateinamen mit der „alten“ DOS Konvention 8.3 verarbeiten. Daher entsprechen die Dateinamen und Einträge auf dem USB Speicher, respektive der Datei „`update.txt`“ dieser Konvention.

Es werden sowohl Dateien, als auch Befehle verarbeitet.

Ab der MFU Nios Software-Version 007.0004 sind auch Kommentare möglich. Kommentarzeilen werden mit einem Hashtag (#) eingeleitet.

```
# Diese Zeile ist ein Kommentar und wird ignoriert.
```

Die Dateiennamen sind dabei frei wählbar, lediglich deren Länge (max. 8.3 Zeichen) darf nicht überschritten werden. Dateien zur Programmierung eines FPGA Application Image enden auf `.rbf` (Raw Binary File). Dateien zur Programmierung der NiosII Application Image Software enden auf `.s19` (Motorola S19).

Im Weiteren besteht die Möglichkeit FW/SW Factory Images und/oder den Advanced Boot Copier zu ersetzten. Deren Programmierdateien müssen die entsprechende Endung aufweisen.

MFU Software Update via USB Flash Device

`.b19`	Advanvced Boot Copier
`.f19`	Factory Software
`.s19`	**Application Software**

FPGA Image Update via USB Flash Drive

`.rfi`	Factory Image
`.rbf`	**Application Image**

4.2.3.4.2. Kommandos

„`update.txt`" kann auch Kommandos enthalten. Diese dienen dazu diverse Initialisierungen während es Programmiervorgangs durchzuführen.

`0 0 *command`

Die Einträge und ihre Bedeutungen:

`usi_number  module_number  *command`

`usi_number`	USI Nummer 1… USI Max (hexadezimal)
`module_number`	Modul Nummer 1… Modul MAX (hexadezimal)
`*command`	das auszuführende Kommando

MFU Kommandos (`usi_number=0, module_number=0`) sind:

`*initoph`	– der MFU internen Betriebsstundenzähler löschen
`*initpar`	– die MFU internen Systemparameter löschen
`*initint`	– die MFU internen Interlocktexte löschen
`*initlog`	– das MFU interne Logbuch löschen
`*initcfg`	– die Netzgerätekonfigurationsdatei löschen
`*initall`	– das gesamte MFU interne serielle Flash löschen

Kommandos für MFU und alle USI Module

`*reconf`	- startet den Wechsel zwischen Factory-Image und Applikation-Image

Wichtig: zwischen `*` und `Command` ist kein Leerzeichen.

4.2.3.4.3. Kommentare

Beginnend mit der MFU Nios Software Version 007.0004 kann die „`update.txt`" auch Kommentarzeilen enthalten. Dies ist praktisch um z.B. verschiedene Firm- und/oder Softwarestände schnell in der „`update.txt`" zu editieren.

Kommentarzeilen werden mit ‚`#`' eingeleitet.

Kommentare sollten nicht länger als 100 Zeichen sein.

4.2.3.4.4. Beispiel

```
#Nachfolgendes programmiert eine neue Firmware in die MFU
0 0 mfu720.rbf
#Nachfolgendes startet die MFU neu im Factory Image
0 0 *reconf
#Nachfolgendes startet die MFU neu im Application Image
0 0 *reconf
#Nachfolgendes programmiert eine neue Firmware ins Modul 1 an USI 3
3 1 icm751.rbf
```

Die mit einem ‚`#`' angeführten Zeilen werden bei der Abarbeitung der „`update.txt`" als Kommentarzeilen ignoriert.

Das `*reconf` Kommando schaltet zwischen den einzelnen Images um. D.h. wurde ein Applikation Image (`.rbf`) ersetzt und soll dieses abschließend gestartet werden, muss das `*reconf` Kommando 2x erfolgen. Beim ersten Mal schaltet das FPGA in das Factory Image, beim zweiten Mal zum neuen Applikation Image.

I.d.R. ist es daher ratsam nach einem Update die Versorgungsspannung zu unterbrechen und das Modul vollständig neu zu starten.

4.3. 2. Schritt manuell - FPGA FW programmieren

Dazu

1. USB Blaster an JTAG anschließen,
2. QuartusII Programmer starten und
3. die Datei `MultiFunctionUnit_FandA.jic` programmieren.

4.4. 3. Schritt manuell - NiosII SW programmieren

4.4.1. MFU_Flash_cfi_flash_ABC_FAC_APP.flash Datei programmieren

4.4.1.1. Was wird dazu benötigt

1. Das NIOSII Terminal.
   Als Teil der NiosII SoftwareBuild Tools installiert, die wiederrum Teil von OuartusII sind.

4.4.1.2. Vorbereitung

1. Ins Installationsverzeichnis der NiosII EDS wechseln (`.../nios2eds/`).
2. Sofern vorhanden dort die Datei „`user.bashrc`“ entfernen oder umbenennen z.B. in „`_user.bashrc`“. Dabei handelt es sich um eine Skriptdatei die automatisch nach dem Start des NiosII Terminals ausgeführt wird.
3. Im Ordner mit Namen „prog`files`“ (`.../nios2eds/progfiles`) wird die evtl. zuvor erstellte Datei `MFU_Flash_cfi_flash_ABC_FAC_APP.flash` gepeichert. Sofern `MFU_Flash_cfi_flash_ABC_FAC_APP.flash` erst erstellt werden muss, siehe Kapitel 5.2: „Erstellen der NiosII SW MFU_Flash_cfi_flash_xxx.flash Dateien für das Parallel-Flash“ auf Seite16.
4. Das Skript „`prog_mfu.sh`“ befindet sich im Verzeichnis „`.../nios2eds`“.
5. Im Skript „`prog_mfu.sh`“ ist der Dateiname des Advanced Boot Copier – Factoy – Applikction Image ggf. entsprechend anzupassen.

4.4.1.3. Der eigentliche Programmiervorgang

1. “`Nios II Command Shell.bat`” starten, dadurch öffnet sich das NiosII Terminal
2. `./prog_mfu.sh` im Terminal eingeben und ENTER drücken
3. Die Datei-Programmierung sollte starten und führt folgende Schritte durch:
   1. Der Parallel-Flash wird vollständig gelöscht.
   2. Das Advanced Boot Copier – Factoy – Application Image wird programmiert.

4.4.1.4. Ein paar warme Wort zu „prog_mfu.sh“

Abbildung 4.4: NiosII Flash Programmer - Optionsfeld

Abhängig von der im NiosII Flash Programmer gewählten Option („Generate File“, „Programm Files“…) wird auch eine Skriptdatei „flash_programmer.sh“ erstellt. Diese wird im QuartusII SOPC Builder NiosII Bibliotheksverzeichnis im Ordner „../script“ abgelegt.

Wird z.B. „Generate Files“ durchgeführt enthält „flash_programmer.sh“ den Shellaufrufe zur Erzeugung der Programmierdatei. Im Falle von „Programm Files“ enthält „flash_programmer.sh“ den Shellaufruf zur Programmierung des Zielsystems. Werden hingegen mehrere Option zeitgleich angewählt und durchgeführt, enthält „flash_programmer.sh“ auch alle zugehörigen Shellaufrufe auf einmal.

Die „prog_mfu.sh“ benötigt den Aufruf zur Programmierung des Zielsystems, die Generierung der Programmierdatei sollte zuvor mittel GUI durchgeführt sein.

Nachfolgend ein Beispiel des Programmieraufrufs. Die einzelnen Flags/Parameter sind hier zur besseren Lesbarkeit durch Return getrennt. Im Aufruf selbst stehen alle ein einer Zeile.

```
$SOPC_KIT_NIOS2/bin/nios2-flash-programmer "MultiFunctionUnit_Flash_cfi_flash.flash"
--base=0x200000
--sidp=0x60A2A8
--id=0x0
--timestamp=1344249066
--device=1
--instance=0 '
--cable=USB-Blaster on localhost [USB-0]'
--program
--erase-all
--go
```

Wichtig ist, dass `--id` und `-timestamp` mit dem NiosII im Zielsystem übereinstimmen.

Die wirklich notwendigen Parameter sind grün markiert.

Seit QuartusII Version 8 führt der NiosII Programmer scheinbar ein Kabel-Auto-Detect durch, so dass die weiteren Parameter den Programmieradapter betreffend weggelassen werden können.

4.5. *4. Schritt manuell VNC2 SW programmieren*

Informationen hierzu finden sich im Dokument ACU-FSP_mUSIc_TFT im Kapitel FSPs MFU FSP255_SW_Flash_VNC2.

4.6. *5. Schritt manuell FTDI Chip Seriennummer programmieren*

Ist eine MFU neu muss die Seriennummer im FT245 von FTDI von einer fortlaufenden Seriennummer in eine einheitliche geändert werden. Andernfalls wird immer wieder ein neues USB Device entdeckt, wenn eine MFU mit dem PC verbunden wird. Eine gleiche Seriennummer in allen MFUs gaukelt dem PC vor, es würde immer das gleiche Geräte angeschlossen.

Leider lässt es sich nicht vermeiden, dass beim Anschluss einer „jungfräulichen“ MFU Windows zunächst einen neuen USB-Seriell Treiber installiert und einen neuen COM-Port vergibt. Nach dem Ändern der Seriennummern ist es ratsam im Gerätemanager die ausgeblendeten Geräte anzuzeigen und die Horde neuer USB-Seriell Converter wieder zu entfernen.

Zum Programmieren wird `FT_Prog` von FTDI und das `USB_FT245R_ACU.xml` Template benutzt.

1. `FT_Prog` starten.
2. MFU via USB Kabel mit dem PC verbinden und warten bis der USB Treiber geladen wurde.
3. F5 (`Devices Scan and Parse`) drücken.
4. Es werden alle gefunden FTDI USB Devices angezeigt.
   ACHTUNG: Auch der Altera USB Blaster enthält einen FDTI Chip. Daher aufpassen, dass nicht ausversehen dieser gelöscht und überschrieben wird.
5. Im Device Tree auf ‚`Device: n`‘ (mit n=1,2,3...) klicken. In Chip Type muss ‚FT245R‘ stehen, dann sollte es das richtige FDTI Device sein.
6. Rechtsklick auf ‚`Device: n`‘ und ‚`Apply Template`‘ -> ‚`From File`‘ -> ‚`USB_FT245R_ACU.xml`‘ auswählen.
7. Rechtsklick auf ‚`Device: n`‘ und ‚`Program Device`‘ auswählen.

8. Der Chip sollte nun mit dem Template programmiert werden.
9. Der PC erkennt die neue Programmierung i.d.R. erst, wenn die USB Verbindung zwischen PC und MFU kurz unterbrochen wird.
   Die Seriennummer lautet nun ‚GSIxxACU'

5. Die Programmierdateien erstellen

5.1. Erstellen der MultiFunctiononUnit_Boot.flash für den EPCS

Diese wird immer dann benötigt, wenn ein „jungfräuliches“ System automatisch neu programmiert werden soll.

Die erzeugte `MultiFunctionUNit_Boot.flash` Datei enthält dabei ein

- MFU FW Boot Image welches ebenfalls eine kleine NiosII Software enthält mit einem Offset von 0x0 im EPCS und ggf. das
- VNC2 Image mit einem Offset von 0x150000 im EPCS

und wird anschließend als Image ins EPCS programmiert. Dadurch ist das MFU FPGA in der Lage zu starten, mittels NiosII ein Programm auszuführen und den VNC2 zu programmieren.

Alle weiteren Programmierschritte können dann über den VNC2 mittels USB Speicher erledigt werden.

5.1.1. Was wird dazu benötigt

1. Das NIOSII Terminal.
   Als Teil der NiosII SoftwareBuild Tools installiert, die wiederrum Teil von OuartusII sind.
2. Das Programm `SRecord Version 1.64`.

5.1.2. Vorbereitung

1. Ins Installationsverzeichnis der NiosII EDS wechseln (`.../nios2eds/`).
2. Sofern vorhanden dort die Datei „`user.bashrc`“ entfernen oder umbenennen z.B. in „`_user.bashrc`“. Dabei handelt es sich um eine Skriptdatei die automatisch nach dem Start des NiosII Terminals ausgeführt wird.
3. `Einen` neuen Ordner mit Namen „`bootfiles`“ anlegen. (`.../nios2eds/bootfiles`).
4. Das Skript „`gen_mfu_boot.sh`“ befindet sich im Verzeichnis „`.../nios2eds`“.
5. Die MFU FPGA Projekt „`.sof`“ Datei befindet sich im Verzeichnis
   „`E:/SVN/FPGAWork/Quartus_111/prj/ACU_MultiFunctionUnit_Boot/ _quartus`“
   Sollte dies nicht der Fall sein, muss das Skript „`gen_mfu_boot.sh`“ in „`.../nios2eds/`“ entsprechend angepasst werden.
6. Die VNC2 Projekt „`.rom`“ Datei befindet sich im Verzeichnis
   „`E:/SVN/CODEWork/prj/VinculumII/V2DAP/Release`“
   Sollte dies nicht der Fall sein, muss das Skript „`gen_mfu_boot.sh`“ in „`.../nios2eds/`“ entsprechend angepasst werden.
7. Das Programm „`srecord`“ befindet sich im Verzeichnis
   „`D:/Programme/srecord-1.64-win32`“
   Sollte dies nicht der Fall sein, muss das Skript „`gen_mfu_boot.sh`“ in „`.../nios2eds/`“ entsprechend angepasst werden.

5.1.3. Der eigentliche Konvertierungsvorgang

1. “`Nios II Command Shell.bat`” starten, dadurch öffnet sich das NiosII Terminal
2. `./gen_mfu_boot.sh` im Terminal eingeben und ENTER drücken
3. Die Datei-Konvertierung sollte starten und führt folgende Schritte durch:
   1. Es wird gefragt, ob auch das VNC2.rom mit in die `.flash` Datei konvertiert werden soll.
   2. Es wird in das MFU FPGA Projektverzeichnis gewechselt und die `MultiFunction-Unit_Boot.sof` Datei ins `.../bootfiles` Verzeichnis kopiert.

3. Es wird aus der `MultiFunctionUnit_Boot.sof` Datei eine `.flash` Datei erzeugt und im Verzeichnis `.../bootfiles` abgelegt.
4. Es wird ggf. ins VNC2 Projektverzeichnis gewechselt und aus der VNC2.rom Datei eine `.flash` Datei erzeugt und im Verzeichnis `.../bootfiles` abgelegt.
   Die erzeigte `.flash` Datei besitzt einen Offset von 0x150000 im EPCS
5. Es wird ins `SRecord` Verzeichnis gewechselt und aus den zwei zuvor erzeugten `.flash` Dateien eine einzige `.flash Datei.`

Im Verzeichnis `.../bootfiles` sollten nun bis zu 5 Dateien zu finden sein:

1. `MultiFunctionUnit_Boot.sof`, eine Kopie der FPGA `MultiFunction-Unit_Boot.sof` Datei aus dem zugehörigen QuartusII Projektverzeichnis.
2. `V2DAP_...flash`, die VNC2 Flash Datei.
3. `MultiFunctionUnit_Boot...flash`, die MFU Flash Datei.
4. `MFU_Boot...flash`, die aus beiden zusammengesetzte Flash Datei.
5. `MFU_Boot...hex`, die ins Intel-Hex Format konvertierte `.flash` Datei.

5.1.4. Ein paar warme Worte zu „gen_mfu_boot.sh“

Kommt die Meldung „`..cannot execute binary..`“ liegt es daran, dass scheinbar das Textformat der „`gen_mfu_boot.sh`“ Datei nicht stimmt, wieso auch immer. I.d.R. sollte dies PC und ANSI sein.

Wie auch die Inhalte der „`prog_mfu.sh`“ lassen sich für „`gen_mfu_boot.sh`“ die notwendigen Einträge mittels NiosII Programmer GUI erzeugen (siehe Kapitel 4.4.1.4: Ein paar warme Wort zu „prog_mfu.sh“) und werden in der Datei „`flash_programmer.sh`“ im QuartusII SOPC Builder NiosII Biblitotheksverzeichnis im Ordner „`../script`“ abgelegt

.

5.2. Erstellen der NiosII SW MFU_Flash_cfi_flash_xxx.flash Dateien für das Parallel-Flash

Diese enthält ausschließlich die NiosII Software:

- den Boot Copier, mit einem Offset von 0x0 im Parallel-Flash
- das Factory-Image, mit einem Offset von 0x10000 im Parallel-Flash
- das Applications-Image, mit einem Offset von 0x100000 im Parallel-Flash

5.2.1. Was wird dazu benötigt

1. Das NIOSII Terminal.
   Als Teil der NiosII SoftwareBuild Tools installiert, die wiederrum Teil von OuartusII sind.

5.2.2. Vorbereitung

1. Ins Installationsverzeichnis der NiosII EDS wechseln (`.../nios2eds/`).
2. Sofern vorhanden dort die Datei „`user.bashrc`“ entfernen oder umbenennen z.B. in „`_user.bashrc`“. Dabei handelt es sich um eine Skriptdatei die automatisch nach dem Start des NiosII Terminals ausgeführt wird.
3. `Einen` neuen Ordner mit Namen „`progfiles`“ anlegen. (`.../nios2eds/progfiles`).
4. Das Skript „`gen_mfu_prog.sh`“ befindet sich im Verzeichnis „`.../nios2eds`“.
5. Die NiosII Flash Projekt „`.elf`“ Datei befindet sich im Verzeichnis „`E:/SVN/CODEWork/prj/NiosIDE/MultiFunctionUnit_Flash/MultiFunction Unit_Flash`“
   Sollte dies nicht der Fall sein, muss das Skript „`gen_mfu_prog.sh`“ in „`.../nios2eds/`“ entsprechend angepasst werden.
6. Der Advanced Boot Copier „abc`.flash`“ Datei befindet sich im Verzeichnis „…`/progfiles`“
7. Soll im nachfolgenden Schritt KEIN Factory-Image erstellt werden, muss sich dieses (`MFU_Flash_cfi_flash_FAC.flash`) schon in „…`/progfiles`“ befinden.

5.3. Der eigentliche Konvertierungsvorgang

1. “`Nios II Command Shell.bat`” starten, dadurch öffnet sich das NiosII Terminal
2. `./gen_mfu_prog.`sh im Terminal eingeben und ENTER drücken
3. Die Datei-Konvertierung sollte starten und führt folgende Schritte durch:
   1. Es wird gefragt, ob ein Factory und Applications-Image erstellt werden soll.
   2. Es wird in das NiosII Flash Projektverzeichnis gewechselt und aus der `NiosIIProject.elf` Datei eine `.flash` Datei erzeugt und im Verzeichnis `.../progfiles` abgelegt.
   3. Es wird ins `SRecord` Verzeichnis gewechselt und die zuvor erzeugten `.flash` Datei mit einem Offset von `0x10000` für ein Application-Image versehen.
   4. Soll ein Factory und Applications-Image erstellt werde, wird zusätzlich eine weitere `.flash` Datei mit einem Offset von `0x100000` erzeugt.
   5. Aus den zuvor erstellten Dateien und der „`abc.flash`“ eine einzige `.flash` Datei erstellen

Im Verzeichnis `.../progfiles` sollten nun bis zu 4 Dateien zu finden sein:

1. `abc.flash`, der Advanced Boot Copier.
2. `MFU_Flash_cfi_flash_APP.flash`, das Application Image.
3. `MFU_Flash_cfi_flash_FAC.flash`, das Factory Image.

4. `MFU_Flash_cfi_flash_ABC_FAC_APP.flash`, das zusammengesetzte Advanced Boot Copier – Factory – Application Image

6. Dinge die der Entwickler wissen MUSS ! ! !

Es gibt scheinbar noch immer ein Problem mit dem NiosII IDE Flash Programmer sofern ein bleifreies EPCS verwendet wird.

In Version 5.1 gab es die Meldung "No EPCS layout data - looking for section [EPCS-202XXX]" wenn der EPCS per JTAG programmiert wurde und bleifrei war.

Angeblich ist der Fehler ab Version 6.0 behoben.

Allerdings war ich auch bei Version 8 und nun auch bei Version 9.0 !! nicht in der Lage diesen zu programmieren. Die Fehlermeldung hieß zwar jetzt:

"No EPCS layout data - looking for section [EPCS-10100A]", trotzdem war folgendes zu tun bis der Mist lief:

Die Datei 'nios2-flash-override.txt' im Verzeichnis nios2eds/bin erstellen mit folgendem Inhalt:

[EPCS-202011] # EPCS1N (lead-free)

sector_size = 32768

sector_count = 4

[EPCS-202013] # EPCS4N (lead-free)

sector_size = 65536

sector_count = 8

[EPCS-10100A] # EPCS16N (lead-free)

sector_size = 65536

sector_count = 32

[EPCS-202017] # EPCS64N (lead-free)

sector_size = 65536

sector_count = 128

Dann funtzt der Kram!

######### Aus dem Nios Forum ############

You will need to create a text file that includes the missing information.

<nios2_install>/bin/nios2-flash-override.txt

and inside this file you include the information like this example

Quote:

[EPCS-010216] # EPCS64N (Spansion-lead-free) S25FL064A0LMFI001

sector_size = 65536

sector_count = 128

The Spansion datasheet for this device says

Manufacture Identification 01h

Memory Type 02h

Memory Capacity 16h

putting all these information together delivers 010216

So you will need to look into the datasheet of your device and find the require information and created the file.

BTW: too bad that the nios forum has lost so many postings, there was a realy complete override file and i remember that epcs device was already listed ...