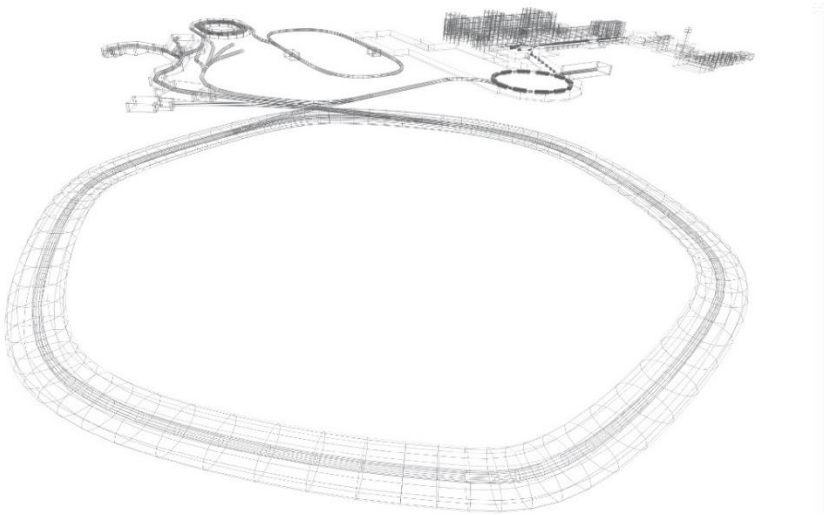
Elektrische
Energie
Technik

GSI Darmstadt



Soft FSPs der Multi Function Unit (Nios)

(ab SW 7.00900)

Version vom: Freitag, 16. August 2024, 10:32:00

Aktuelle Versionen immer unter:

https://wiki.gsi.de/EPS/ACUModuleDescriptions

Inhaltsverzeichnis

$\mathbf{0xFF_H/255_D/0x4646_{ASCII}}$

1. Änderungsliste

Datum	Name	Kommentar
16.08.2024	D. Schupp	Dokument erstellt aus ACU-FSP MFU_SE

2. Soft FSPs MFU (SE/LE/LWL)

Dieses Dokument behandelt modulspezifische Soft FSPs der MultiFunctionUnit (MFU).

FSP beginnend mit „_SW_“ sind nicht in der MFU Hardware implementiert, sondern Bestandteil der Nios-Software.

Name	**FSP224_SW_CUSTOMIZED_PULSES_PER_LITER**
Adresse	**$0xE0_H/224_D/0x4530_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Beinhaltet Daten die zur Anzeige von benutzerdefinierten Durchflusswächtern auf dem TFT der MFU nötig sind.

Verfügen Hardware-Module über die Funktion zum Anschluss von Durchflusswächtern und werden diese benutzerdefiniert, reichen die Informationen des Hardware FSP055 im Modul selbst nicht aus, um den aktuell gemessenen Durchfluss auf dem TFT der MFU darzustellen. Die hierzu notwendige [PulsesPerLiter] Information wird in diesem FSP abgelegt.

Lesen/Schreiben

`STX PID PID GW MA FSP FSP`

[schreiben: für jedes Modul wie folgt:

`USINr.`[1 Byte ASCII] `ModulNr.`[1 Byte ASCII] `Anzahl Durchflusswaechter` [2 Byte ASCII]

`Durchflusswaechter(n)` [4 Byte ASCII]

`Durchflusswaechter(n-1)` [4 Byte ASCII]

`Durchflusswaechter(n-2)` [4 Byte ASCII]

`Durchflusswaechter(n-3)` [4 Byte ASCII]

`Durchflusswaechter(n-..)` [4 Byte ASCII]

Dann wiederholt sich das Ganze für evtl. weitere Module.

`PP PP`

]

`ETX`

Beim Lesen werden die Daten entsprechend zurück geliefert.

Name	**FSP225_SW_READ_RECORDED_MODULE_DATA_FROM_DATA_STORAGE**
Adresse	**$0xE1_H/225_D/0x4531_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	0x(siehe Beschreibung)$_H$

Für die Nutzung von FSP225 muss das datensammelnde Modul über ein [DataStorage] Modul verfügen.

Über diesen FSP werden die gesammelten Daten des [DataStorage] via HighSpeed an die MFU und von dieser weiter an den angeschlossenen Computer übertragen.

Wichtig ist, die zu lesenden Daten müssen schon im [DataStorage] gespeichert sein.

Die Konfiguration von [DataStorage] ist explizit nicht Bestandteil dieses FSP und muss zuvor extern erfolgen.

Die externe Sequenz zur Aufzeichnung der Daten in [DataStorge] ist wie folgt:

1) FSP225 zunächst schreiben:
   `STX PID PID GW MA FSP FSP [USI Number] PP PP ETX (11 Byte)`

   Parameter [ModuleNumber] ist nicht nötig, da Modul Nummer bei HighSpeed uninteressant
   => immer Modul 1(0).

2) `Reset` Kommando setzen/löschen in Modul FSP018[0].
3) Status lesen (Modul FSP017) und auf `0x00` testen.
4) `ExtTriggerCommand` setzen (active high) (Modul FSP018[8]).
5) Status lesen (Modul FSP017), muss nun `0x05` sein (Ereignis getriggert, Werte erfasst).
6) `ExtTriggerCommand` loeschen (low) (Modul FSP018[8]).
7) Status lesen (Modul FSP017), muss nun `0x01` sein (Werte erfasst).
8) `RdEnable` setzen (Modul FSP018[4]).

Jetzt kommt das, was diese Funktion tut:

1) Kommando `CMDTriggerSomething` absetzen durch setzen/löschen von `CPU_STATUS_CMD_TRIGGER_SOMETHING` im CPU_STATUS[30].
2) Aufgezeichnete Messwerte per HighSpeed empfangen.
   Die Antwort ist dabei wie folgt:

```
STX GW MA FSP FSP  [Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] <- n
                   [Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] <- n-1
                   [Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] <- n-2
                   [Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] <- n-3
                   [Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] <- n-4
                   [...]
                   [Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] <- n-n
PP PP ETX
```

   Die Anzahl (n) der gesendeten Datensätze ist abhängig von der [DataStorage] Implementierung.

Im Anschluss extern:

1) Status lesen (FSP017), muss nun `0x03` sein

Name	**FSP226_SW_FSP_FILL_FGSTIMULI_RAM**
Adresse	**$0xE2_H/226_D/0x4532_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

FSP226 ermöglicht das RAM für den FG-Stimuli mit Daten zu füllen.

Es werden die Daten einer `.hex`-Datei ins RAM geschrieben.

```
:0C 0000 00 57170D57170C81446F860706 38
:0C 0001 00 57170D550D1C8290A8610606 D3
||  |    |  |                       \-- Pruefsumme
||  |    |  \-------------------------- Daten (13 Byte)
||  |    \----------------------------- Typ
||  \---------------------------------- Adresse
|\------------------------------------- Laenge der Daten (13 Byte)
\-------------------------------------- Start
```

Das RAM des FG-Stimuli hat eine max. Tiefe von 4096 x 96Bit.

Um den Datenbus des NIOS nicht unnötig aufzublähen, werden die Daten zu je 6-Bit ins RAM geschrieben.

Also jeweils 16 x 6 Bit mit 16 Adressen anstelle 1 x 96 Bit mit einer Adresse.

D.h.
```
96-Bit Adresse 0x0000 <=> 6-Bit Adressen 0x0000 ... 0x000F
96-Bit Adresse 0x0001 <=> 6-Bit Adressen 0x0010 ... 0x001F
96-Bit Adresse 0x0002 <=> 6-Bit Adressen 0x0020 ... 0x002F
usw.
```

Seitens der Hardware werden diese Daten aber mit 96 aus dem RAM ausgelesen.

Schreiben des FSP:

```
STX PID PID GW MA FSP FSP [.hex-Datei] PP PP ETX
```

NIOS Interface
==========

```
FG_RAM_NmbOfDataTupels_Pio(16)
```

[15..12] 0000

[11..0] NmbOfDataTupels(12 Bit)

```
FG_RAM_Ctl_Addr_Data_Pio  (32)
```

[31] Enable(1 Bit),
wenn '1' werden die o-Ausgänge des "ACU_FG_Stimuli" an "fg_quad_scu_bus" gelegt, andernfalls die i-Eingänge des "ACU_FG_Stimuli".

[30] FGEnable(1 Bit),
Entspricht Bit[1] des FG Ctrl Registers, starten/stoppen des FG.

[29] RAM_WrEnable(1 Bit),
Wenn '1' werden Daten an den Eingängen "RAM..." ins RAM übernommen.

[28] FGReset(1 Bit),
Wenn '1' erfolgt in 'ACU_FG_Stimuli' UND 'fg_quad_scu_bus' ein Reset.

[27..24] (0000)

[23..8] RAM_WrAddress(16 Bit),
Adresse (zusammen mit 'RAM_WrEnable').

[7..6] (00)

[5..0] RAM_WrData(6 Bit),
Daten (zusammen mit 'RAM_WrEnable')

Name	**FSP227_SW_FSP_BIT_MAN_ENHANCED**
Adresse	**$0xE3_H/227_D/0x4533_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

FSP227 ermöglicht Bitmanipulationen in beliebigen FSPs, beliebiger Module.

Basiert auf dem [FSP241_SW_BitManipulation] (siehe Seite 21), der (leider) nur MFU interne FSPs bedienen kann.

Einschränkung: es können nur die Bits zwischen 0..255 manipuliert werden.

Aufbau der Anforderung

`STX PID PID GW MA FSPhi FSPlo USINrTarget ModuleNrTarget FSPmHi FSPmLo BITPosHi BITPosLo BITVal PP PP ETX` (17 Byte)

`STX`	StartOfText (0x02)
`PID`	Schreibanforderung (0x57 0x52)
`GW`	Gateway Adresse (0x30 da Gateway die MFU ist)
`MA`	Modul Adresse (0x30 da Modul die MFU ist)
`FSPhi FSPlo`	FSP Nummer (0x45, 0x33)
`USINrtarget`	USI an der das Modul zu finden ist (0x30(MFU), 0x31..0x0A)
`ModuleNrTarget`	Modul Nummer an diesem USI (0x30(MFU), 0x31..0x38)
`FSPmHi FSPmLo`	zu manipulierendes FSP (1...255)
`BITPosHi BITPOsLo`	Bitposition (0...255, gibt die Bitposition an)
`BITVal`	Wertigkeit des Bit (gleich 0(0x30) -> Bit löschen, ungleich 0 -> Bit setzen)
`PP`	Prüfsumme (Prüfsumme der Daten in Hex als 2 Byte ASCII)
`ETX`	EndOfText (0x03)

Name	**FSP228_SW_SCOPE_HEAD**
Adresse	**$0xE4_H/228_D/0x4534_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

FSP228 ermöglicht die Umschaltung der Soll-/Istwertanzeige auf dem TFT.

Experimentell: noch nicht vollständig implementiert.

Name	**FSP229_SW_HighSpeedStream_Synchronized**
Adresse	**$0xE5_H/229_D/0x4535_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(siehe\ Beschreibung)_H$

FSP229 basiert auf dem „FSP237_SW_HighSpeedStream“ und gibt ebenfalls eine bestimmte Anzahl von Daten aus, die über einen wählbaren HighSpeed Kanal gesammelt werden.

Bedingung hierbei ist, dass das Zielmodul über diese Funktionalität verfügt. Dazu müssen im Zielmodul die FSP017 und FSP018 vorhanden sein.

Die Unterschiede dabei sind:

- die Datenerfassung ist synchronisiert, d.h. die erfassten Daten werden nicht nach einer voreinstellbaren Zeit gepollt, sondern es wird jeder neue Messwert erfasst
- die Anzahl lesbarer HS Datensätze ist größer 2^{16} (=> max. 2^{32}), ist wegen des begrenzten RAM im Modul/der MFU aber nicht nutzbar). Dabei ist die Anzahl lesbarer Messwerte im Modul FSP017 zu finden und wird nicht vom Nutzer gesetzt
- vor dem Lesen wird
  1.) der Regler gesperrt
  2.) im auszulesenden Modul seitens der MFU eine Initialisierung für den Ausleseprozess gestartet.

Bevor das FSP gelesen werden kann, muss es beschrieben werden mit:

```
STX PID PID GW MA FSP FSP [USINr.] PP PP ETX
```

Mit:

`PID`	WR
`[USINr.]`	USI Nummer (0x01..0x0B)

Im Anschluss wird das FSP gelesen.

Die MFU führt daraufhin die folgenden Aktionen aus:

- sämtliche USIs seitens der MFU werden deselektiert
- es wird geprüft ob die aktuelle MFU Firmware diese Funktion unterstützt
- FSP017 des Zielmoduls lesen. Dort ist die Anzahl der erfassbaren Messwerte zu finden.
- ist FSP017 des Zielmoduls lesbar. die passende Speichergröße in der MFU reservieren
- den Regler sperren
- im Zielmodul FSP018[0] einen Reset der Datenerfassung ausführen
- im Zielmodul FSP018[2] den Trigger zur Datenerfassung aktivieren
- nun wird die Anzahl von Daten laut FSP017 des Zielmoduls, im Zielmodul erfasst und auch dort gespeichert
- FSP017 des Zielmoduls lesen bis Bit[1] gesetzt ist (Messung beendet)
- Zielmodul FSP018[1] setzen, dies ermöglicht das lesen der Messdaten
- die MFU setzt das Kommando „TriggerSometing“ ab, dadurch startet das Zielmodul den Datentransfer der Messwerte über den HighSpeed Kanal. Dazu wird der „normale HighSpeed Stream“ unterbrochen. Stattdessen findet der Messwertdatentransfer statt.
- sind alle Daten übertragen werden diese an die übergeordnete Instanz (Kontrollsystem/Anwender PC) ausgegeben
- der „normale HigHSpeed Stream“ wird wieder auf der USI aktiviert

Die Daten sehen dabei wie folgt aus:

```
STX GW MA FSP FSP
[Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] ← Datensatz n
[Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] ← Datensatz n-1
[..]
[Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] ← Datensatz n-n
PP PP ETX
```

Name	**FSP230_SW_intScopeHeaderReadOut**
Adresse	**$0xE6_H/230_D/0x4536_{ASCII}$**
Tiefe	6 Byte / 48 Bit
I/O	lesen
Reset	0x(siehe Beschreibung)$_H$

Der interne SW FSP230 ist ein CPU Software FSP und stellt die TimeScale, TimeBase, TriggerOffsetEcho, und TriggerAddress des internen Oszilloskops über den USB→PC Anschluss zur Verfügung.

Diese Daten werden von PCA (PowerConfigAdvanced) zur Darstellung der internen Oszilloskopdaten benötigt.

FSP230 kann nur gelesen werden. Schreibanforderungen bekommen eine NACK Antwort mit Fehlercode.

Die Informationen des FSP230 werden dabei aus Inhalten der „**Fehler! Verweisquelle konnte nicht gefunden werden.**“ und „**Fehler! Verweisquelle konnte nicht gefunden werden.**“ gewonnen.

Ausgabestring nach der Leseanforderung:

```
STX GW MA -FSP-  -TS--  ----TB-----  ----TOE---- ---TA--- --PP-  ETX
02  30 30 45 36  ts ts  tt tt tt tt  toe toe toe ta ta ta pp pp  03
```

TS	TimeScale	Eingestellte Zeitskalierung für TimeBase des internen Oszilloskops (2 Byte ASCII = 8 Bit HEX)
TB	Timebase	Eingestellte Zeitbasis des internen Oszilloskops (4 Byte ASCII = 16 Bit HEX)
TOE	TriggerOffsetEcho	Triggeroffset vom Nullpunkt der X-Achse (3 Byte ASCII = 12 Bit HEX)
TA	TriggerAddress	Speicheradresse an der das Triggerereignis stattgefunden hat (3 Byte ASCII = 12 Bit HEX)

Der Speicher des int. Oszilloskops ist als Ringspeicher ausgeführt. D.h. solange kein Triggerereignis erfolgt werden die Daten zyklisch in diesen Ringspeicher eingetragen.

Findet nun ein Triggerereignis statt, wird in `TriggerAddress` die Adresse gespeichert an der das Ereignis stattgefunden hat, `TriggerOffsetEcho` gibt die Trigger Offsetverschiebung vom Nullpunkt der X-Achse an.

Name	**FSP231_SW_intScopeDataStreamReadOut**
Adresse	**$0xE7_H/231_D/0x4537_{ASCII}$**
Tiefe	12008 Byte / 96064 Bit (bis MFU FW 7.4.x) 6008 Byte / 48064 Bit (ab MFU FW 7.5.0)
I/O	lesen
Reset	$0x(\text{siehe Beschreibung})_H$

Der interne FSP231 ist ein CPU Software FSP und stellt die eigentlichen Daten des internen Oszilloskops als Stream über den USB→PC Anschluss zur Verfügung.

Diese Daten werden von PCA (PowerConfigAdvanced) zur Darstellung der internen Oszilloskopdaten benötigt.

FSP231 kann nur gelesen werden. Schreibanforderungen bekommen eine NACK Antwort mit Fehlercode.

Sollen die Daten beginnend vom Triggeroffset und nicht vom RAM Speicher 0 ausgegeben werden bitte den FSP245_SW_intScopeDataStream benutzen.

Bis MFU FW 7.4.x gilt folgendes:

Ausgabestring nach der Leseanforderung:

```
STX GW MA -FSP-  -CH8(12Bit) CH7(12Bit) CH7(12Bit) CH5(12Bit)
CH4(12Bit) CH3(12Bit) CH2(12Bit) CH1(12Bit)- -PP-   ETX

02  30 30 45 37   dd dd dd   dd dd dd   dd dd dd   dd dd dd
dd dd dd   dd dd dd   dd dd dd   dd dd dd    pp pp  03
```

CHx - Daten des Kanals

Insgesamt werden 500 x **8** Kanäle übertragen. D.h. der gelb markierte Bereich wird also insgesamt 500-mal übertragen. Beginnend mit dem Wert 1 für Kanal 8 bis 1, Wert 2 für Kanal 8 bis 1 usw. Die Prüfsumme PP folgt nach Wert 500 für Kanal 8 bis 1 über die gesamten Daten.

Ab MFU FW 7.5.0 gilt folgendes:

Ausgabestring nach der Leseanforderung:

```
STX GW MA -FSP-  -CH4(12Bit) CH3(12Bit) CH2(12Bit) CH1(12Bit)- -PP-
ETX

02  30 30 45 37   dd dd dd   dd dd dd   dd dd dd   dd dd dd    pp pp
03
```

CHx - Daten des Kanals

Insgesamt werden 500 x **4** Kanäle übertragen. D.h. der gelb markierte Bereich wird also insgesamt 500-mal übertragen. Beginnend mit dem Wert 1 für Kanal 4 bis 1, Wert 2 für Kanal 4 bis 1 usw. Die Prüfsumme PP folgt nach Wert 500 für Kanal 4 bis 1 über die gesamten Daten.

Name	**FSP232_SW_intSystemParameters**
Adresse	**$0xE8_H/232_D/0x4538_{ASCII}$**
Tiefe	dynamisch
I/O	lesen
Reset	$0x(\text{siehe Beschreibung})_H$

Der interne FSP232 ist ein CPU Software FSP und stellt die ACU Systemparameter als Stream über den USB→PC Anschluss zur Verfügung.

FSP232 kann nur gelesen werden. Schreibanforderungen bekommen eine NACK Antwort mit Fehlercode.

Nach einer Leseanforderung werden die in der MFU gespeicherten Systemparameter übertragen. Diesen wird der reguläre USI Header vorangestellt und der Stream mit Prüfsumme und ETX abgeschlossen.

Wichtiger Hinweis:
Die Systemparameter sind als USI konforme Strings im MFU Speicher hinterlegt. d.h. jeder dieser Einzelstrings wird mit einem STX [0x02] eingeleitet und einer Prüfsumme über die Daten dieses Einzelstrings, sowie einem ETX[0x03] abgeschlossen. Die Summe aller Systemparameter wiederrum wird ebenfalls als USI konformer String ausgegeben. Dieser wird also ebenfalls durch ein ETX eingeleitet und mit Prüfsumme (diesmal über alle vorherigen Parameterstrings) und ETX abgeschlossen.

□	0	0	E	8	□	W	R	0	0	F	1	0	D	0	1	0	0	7	5	□	□	W	R
02	30	30	45	38	02	57	52	30	30	46	31	30	44	30	31	30	30	37	35	03	02	57	52

Einleitung der FSP Antwort mit STX, Gateway, Moduladresse, FSP Nummer ($E8_H = 223_D$)
erster Parameterstring
Anfang des zweiten Parameterstring
es folgen ggf. weitere Parameterstrings

0	0	0	0	0	0	0	□	4	C	□
30	30	30	30	30	30	30	03	34	43	03

Rest des letzten Parameterstring
Abschluss der FSP Antwort mit Prüfsumme (über alle Parameterstrings, inkl. deren STX, „WR“, Gateway, Modulnummer, Daten, Prüfsummen und ETX) und ETX

Das Speichern der Systemparameter wird mittels Bit[7] im „FSP242_SW_CPU_Status“ (siehe Seite 22) eingeleitet und auch bestätigt.

Name	**FSP233_SW_InterlockTexts**
Adresse	**$0xE9_H/233_D/0x4539_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Der interne FSP233 ist ein CPU Software FSP und beinhaltet die für das ACU System in der MFU hinterlegten Interlocktexte.

Weitere Informationen zum beschreiben dieses FSP finden Sie im Dokument "ACU-MFU-FSP233-Interlocktexte programmieren".

Nach einer Leseanforderung werden die in der MFU gespeicherten Interlocktexte übertragen. Diesen wird der reguläre USI Header vorangestellt und der Stream mit Prüfsumme und ETX abgeschlossen.

Name	**FSP234_SW_MDS**
Adresse	**$0xEA_H/234_D/0x4541_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Der interne FSP234 ist ein CPU Software FSP und liefert die MDS der an den USI gefundenen Module im ACU System.

Dem voran wird die MDS der MFU selbst gesendet.

Wird FSP234 geschrieben, startet dies einen USI Rescan. In diesem Fall werden geschickte Dummy-Daten und Prüfsumme nicht ausgewertet.

```
STX PID PID GW MA FSP FSP Data Data PP PP ETX
0x02 0x57 0x52 0x30 0x30 0x45 0x41 0x30 0x30 0x30 0x30 0x03
```

Nach einer Leseanforderung werden die im ACU System gefundenen Moduldeskriptoren wie folgt ausgegeben:

				Datenbytes für Prüfsumme												
Direkte Antwort des FSP234				MDS der MFU					MDS gefundener Module						Prüf-summe	
STX	GW	MA	FSP	USINr. (GW)	MA	Mod. FSP	MDS ACU	PP	USINr.	MA	Mod. FSP	MDS Modul	PP		PP	ETX
0x02	0x30	0x30	0x4541	0x30	0x30	0x3030	(...EOD)		0x3u	0x3m	0x3030	(...EOD)			0x.. 0x..	0x03
Einleitung mit STX, der Gateway- und Modul-, sowie die FSP Nummer				Die MDS der MFU wird wie die nachfolgenden Modul MDS mit der USI Nummer (in diesem Falle die Gatewaynummer 0, da es sich ja um die MFU handelt), der Modul- und FSP Nummer eingeleitet. Der Deskriptor wird mit dem End-deskriptor (…EOD) und der an-schließenden Prüfsumme (PP = 2 Byte ASCII) abgeschlossen. Es folgt KEIN ETX!					Die MDS der an der MFU ange-schlossenen und identifizierten Module. 'u' bei USINr. liegt dabei zwischen 1...10, 'm' bei Mod.Nr. zwischen 1...8. Die einzelnen Deskriptoren werden mit dem Enddeskriptor (…EOD) und der anschließenden Prüf-summe (PP = 2 Byte ASCII) abgeschlossen, es folgt erst nach der letzten MDS ein ETX!					2 Byte ASCII Prüf-summe über alle Daten-bytes	ETX nach der letzten gesende-ten MDS.

Name	**FSP235_SW_Logbook**
Adresse	**$0xEB_H/235_D/0x4542_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Der interne FSP235 ist ein CPU Software FSP und liefert das MFU interne Logbuch.

Jeder Eintrag des Logbuchs ist 8 Byte lang.
Die ersten 5 Bytes beinhalten das Datum und die Uhrzeit des Eintrags.
Die RTC liefert die Daten in BCD.
Im Logbuch sind diese aber aus Platzgründen in komprimiert in HEX abgelegt

Bezeichnung	Bereich	Bitdarstellung	Kurzformat
Tag	1..31	5 Bit 00001..11111	D
Monat	1..12	4 Bit 0001..1100	M
Jahr	0..99	7 Bit 0000000..1100011	Y
Stunde	0..23	5 Bit 00000..10111	H
Minute	0..59	6 Bit 000000..111011	m
Sekunde	0..59	6 Bit 000000..111011	S
Wochentag	0..6	3 Bit 000.110 (0 = Sonntag)	W

Die obige BCD Darstellung wird zur Ausgabe in die nachfolgende HEX Darstellung transferiert.

Byte	Inhalt
0	DDDD DMMM
1	MYYY YYYY
2	HHHH Hmmm
3	mmmS SSSS
4	SWWW xxxx

Die drei nachfolgenden Bytes enthalten den eigentlichen Logbucheintrag.

Byte	Inhalt	Details
5	Spezifikation	0 = normaler Eintrag
		1 = Fehler
		2 = Warnung
		3 = Interlock
6	Eintrag MSB	Nibbel 3..2
7	Eintrag LSB	Nibbel 1..0

Normal:

Nibble	Bedeutung	Bereich
3	Logbucheintrag	$(0..255)_D$
2		
1		
0		

Fehler:

Nibble	Bedeutung	Bereich
3	USI Nummer	$0x0..0xB\ (0..11)_D$
2	Modul Nummer	$0x0..0x7\ (0..7)_D$
1	Fehlernummer	0x00...0xFF (1.127)
0		

Warnung:

Nibble	Bedeutung	Bereich
3	Bitposition der Warnung in Dezimal	$(0..31)_D$
2		
1		
0		

Interlockeintrag:

Nibble	Bedeutung	Bereich
3	USI Nummer	$0x0..0xB\ (0..11)_D$
2	Modul Nummer	$0x0..0x7\ (0..7)_D$
1	Interlockbit	0x00...0xFF (0.255)
0		

FSP235 kann nur mit einem legitimierten USB Stick geschrieben werden. Andernfalls bekommen Schreibanforderungen eine NACK Antwort mit Fehlercode.

Ist ein legitimierter USB Stick eingesteckt und wird FSP235 geschrieben, wird das Logbuch gelöscht. In diesem Fall werden geschickte Dummy-Daten und Prüfsumme nicht ausgewertet.

STX PID PID GW MA FPS FSP Data Data PP PP ETX

0x02 0x57 0x52 0x30 0x30 0x45 0x42 0x30 0x30 0x30 0x30 0x03

Name	**FSP236_SW_Delete_Errors**
Adresse	**$0xEC_H/236_D/0x4543_{ASCII}$**
Tiefe	dynamisch
I/O	schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Der interne FSP236 ist ein CPU Software FSP und löscht den MFU internen Fehlerspeicher.

In diesem Fall werden geschickte Dummy-Daten und Prüfsumme nicht ausgewertet.

```
STX PID PID GW MA FPS FSP Data Data PP PP ETX
0x02 0x57 0x52 0x30 0x30 0x45 0x43 0x30 0x30 0x30 0x30 0x03
```

Name	**FSP237_SW_HighSpeedStream**
Adresse	**$0xED_H/237_D/0x4544_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Gibt eine bestimmte, frei wählbare Anzahl von USI HighSpeed Daten mit einer ebenfalls frei wählbaren Zeitverzögerung zurück.

Dazu muss der FSP zunächst beschrieben werden mit:

```
STX PID PID GW MA FSP FSP
[USINr.][Quantity][Quantity][Quantity][Quantity][Pause/Skip] PP PP ETX
```

Mit:

`PID`	WR
`[USINr.]`	USI Nummer (0x1..0xB)
`[Quantity]`	Anzahl von Werten ‚n' (1..65536)
`[Pause/Skip]`	Wartezeit zwischen zwei Werten (0..15) in uSekunden, bzw. bei neuer MFU FW die Anzahl von Werten die zwischen zwei erfassten Werte ignoriert werden sollen. Wird der FSP anschließend gelesen, werden ‚n' HighSpeed Datensätze gesendet:

```
STX GW MA FSP FSP
[Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] ← Datensatz n
[Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] ← Datensatz n-1
[..]
[Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0] ← Datensatz n-n
PP PP ETX
```

Ab MFU SE FW 7.4.0, bzw. MFU LE FW 7.3.0 ist die Datenerfassung synchronisiert, d.h. die erfassten Daten werden nicht mehr nach einer voreinstellbaren Zeit gepollt, sondern es wird jeder neue Messwert erfasst. `[Pause]` gibt jetzt nicht mehr die Zeitdauer zwischen zwei erfassten Messwerten in uSekunden an, sondern wird zu `[Skip]` und bestimmt die Anzahl von Messwerten die zwischen zwei erfassten Messwerten verworfen werden sollen.

Name	**FSP238_SW_SnapshotHighSpeed**
Adresse	**$0xEE_H/238_D/0x4545_{ASCII}$**
Tiefe	dynamisch
I/O	lesen
Reset	$0x(\text{siehe Beschreibung})_H$

Liefert einen Schnappschuss der Highspeed Daten aller aktiven und im HighSpeed Modus befindlichen USIs (1..11) zurück.

Eine komplette Antwort lautet:

```
STX GW MA FSP FSP
[USI01Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI02Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI03Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI04Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI05Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI06Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI07Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI08Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI09Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI10Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
[USI11Nr. Byte7 Byte6 Byte5 Byte4 Byte3 Byte2 Byte1 Byte0]
PP PP ETX
```

mit:

`[ ]` sofern diese USI aktiv ist. Andernfalls wird die entsprechende Zeile bei der Ausgabe übersprungen.

Name	**FSP239_SW_Debug**
Adresse	**$0xEF_H/239_D/0x4546_{ASCII}$**
Tiefe	65536 Byte / 524288 Bit
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Dieser FSP liefert Inhalte von Flashsektoren zurück -> für Debugzwecke

Wird der FSP beschrieben, muss der zu lesende Sektor mitgeteilt werden.

```
STX GW MA FSP FSP SectorNr.High SectorNr.Low PP PP ETX
```

Beipiel: Sektor 5 lesen

□	W	R	0	0	E	F	0	5	0	5	□
02	57	52	30	30	45	46	30	35	30	35	03

Wird der FSP gelesen liefert er die Daten des zuvor gewählten Sektors.

Nach dem Bootvorgang der MFU ist immer der Sektor 5 (Parameter) gewählt.

Name	**FSP240_SW_RealTimeClock**
Adresse	**$0xF0_H/240_D/0x4630_{ASCII}$**
Tiefe	7 Byte / 56 Bit
I/O	Lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Der interne FSP240 ist ein CPU Software FSP und dient dem Stellen und Auslesen der MFU internen RTC (Real Time Clock).

FSP240 kann geschrieben und gelesen werden.

Zum Beschreiben wird eine reguläre USI Schreibanforderung mit folgender Syntax an dieses FSP gesendet,

```
STX PID PID GW MA FSPhi FSPlo wDay wDay Day Day Month Month Year Year
Hour Hour Minute Minute Second Second PP PP ETX
```

OLD_RTC (RTC im FPGA)

[55..48]	`wDay`, Wochentag	(0x00..0x06, wobei Sonntag = 0)
[47..40]	`Day`, Tag	(0x01..0x1F → 1..31))
[39..32]	`Month`, Monat	(0x01..0x0C → 1..12)
[31..24]	`Year`, Jahr	(0x00..0x63 → 0..99)
[23..16]	`Hour`, Stunden	(0x00..0x17 → 0..23)
[15..8]	`Minute`, Minuten	(0x00..0x3B → 0..59)
[7..0]	`Second`, Sekunden	(0x00..0x3B → 0..59)

Jeder Einstellwert (Tag, Monat usw.) besteht aus 2 Byte ASCII mit hexadezimaler Information. Nicht benötigte Bytes wie das MSB beim Wochentag und Monat werden als 0x30 gesendet.

Wird der FSP gelesen wird die Zeitinformation in der gleichen Reihenfolge ausgegeben.

Die Millisekunden werden weder von extern geschrieben noch ausgegeben. Nach dem senden der Uhrzeitdaten vom PC werden die Millisekunden auf 0 gesetzt und die RTC gestartet.

NEW_RTC (RTC im eigenen Chip, FPGA extern)

Alle Daten sind BCD codiert.

[55..48]	`wDay`, Wochentag	(0x30, 0x30 .. 0x30, 0x36, → 0 .. 6, mit Sonntag = 0)
[47..40]	`Day`, Tag	(0x30, 0x31 .. 0x33, 0x31 → 1 .. 31))
[39..32]	`Month`, Monat	(0x30, 0x31 .. 0x31, 0x32 → 1 .. 12)
[31..24]	`Year`, Jahr	(0x30, 0x30 .. 0x39, 0x39 → 0 .. 99)
[23..16]	`Hour`, Stunden	(0x30, 0x30 .. 0x32, 0x33 → 0 .. 23)
[15..8]	`Minute`, Minuten	(0x30, 0x30 .. 0x35, 0x39 → 0 .. 59)
[7..0]	`Second`, Sekunden	(0x30, 0x30 .. 0x35, 0x39 → 0 .. 59)

Jeder Einstellwert (Tag, Monat usw.) besteht aus 2 Byte ASCII mit BCD Information. Das nicht benötigte MSB Byte beim Wochentag wird immer als 0x30 gesendet.

Wird der FSP gelesen wird die Zeitinformation in der gleichen Reihenfolge ausgegeben.

Erst nach dem erfolgreichen senden aller Uhrzeitdaten vom PC wird die RTC gesetzt und ggf. gestartet.

Im folgenden Beispiel wird die RTC auf 16:15:25 20.06.2012 gestellt.

□	W	R	0	0	F	0	0	3	2	0	0	6	1	2	1	6	1	5	2	0	0	5	□
02	57	52	30	30	46	30	30	33	32	30	30	36	31	32	31	36	31	35	32	30	30	35	03

Name	**FSP241_SW_BitManipulation**
Adresse	**$0xF1_H/241_D/0x4631_{ASCII}$**
Tiefe	3 Byte / 24 Bit
I/O	schreiben
Reset	0x(siehe Beschreibung)$_H$

Der interne FSP241 ist ein CPU Software FSP und dient der Bitmanipulation der FSP in der MFU.

Für Bitmanipulationen in beliebigen FSPs, beliebiger Module, benutzen Sie bitte den [FSP227_SW_FSP_BIT_MAN_ENHANCED], (siehe Seite 6).

FSP241 kann nur geschrieben werden. Leseanforderungen bekommen eine NACK Antwort mit Fehlercode.

Zum Beschreiben wird eine reguläre USI Schreibanforderung mit folgender Syntax an dieses FSP gesendet,

```
STX PID PID GW MA FSPhi FSPlo [FSPmhi FSPmlo] [BIThi BITlo] [BITVal
BITVal] PP PP ETX
```

Einschränkung: es können nur die Bits 0..255 manipuliert werden.

[23..16] `FSPmhi/FSPmlo`, zu manipulierendes FSP (nur beschreibbare FSP)

[15..8] `BIThi/Bitlo`, Bitnummer (gibt die Bitposition an, beginnt bei 0…255)

[7..0] `BITVal`, Wertigkeit des Bit,
gleich = 0x00 → Bit löschen,
ungleich ! = 0x00 → Bit setzen

Name	**FSP242_SW_CPU_Status**
Adresse	**$0xF2_H/242_D/0x4632_{ASCII}$**
Tiefe	4 Byte / 32 Bit
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Der interne FSP242 ist ein CPU Software FSP und repräsentiert 'CPU_Status' Informationen.

Einige Bit können von extern geschrieben und damit bestimmte Funktionen aktiviert werden.
[7] 'CPU_STATUS_RECEIVE_SYSPARAMETERS'
[31] 'CPU_STATUS_DISABLE_CIRCULAR_INTERLOCK_CHECK'.
Alle übrigen Bit sind **Read Only**.

Bit	Bezeichnung	R/W	Beschreibung
[31]	CPU_STATUS_DISABLE_CIRCULAR_INTERLOCK_CHECK	R/W	Wenn [1] wird KEINE zyklische Interlockabfrage durchgeführt. TIMER_INTERLOCK_CIRCULAR_CHECK ist gestoppt.
[30]	CPU_STATUS_CMD_TRIGGER_SOMETHING	R	Wenn [1] wird das Kommando '`cCMD-TiggerSomething`' von 'inst_MFUSwitchingOperations' ausgegeben.
[29]	CPU_STATUS_DISABLE_ALL_TIMERS	R/W	Wenn [1] sind alle Hardware Timer gestoppt: TIMER_STD_SCREEN_UPDATE TIMER_WARNING_UPDATE TIMER_INTERLOCK_CIRCULAR_CHECK TIMER_RECORD TIMER_RETURN_TO_STD_SCREEN TIMER_TFT_DIM_BACKLIGHT TIMER_UNIVERSAL
[28]	CPU_STATUS_FG_STIMULI_ENABLED	R/W	Wenn [1] „füttert“ FG Stimuli den Funktionsgenerator mit Daten.
[27..21]		R	n.u.
[20]	CPU_STATUS_SYSTEM_HAS_INTERLOCKS	R	Wenn [1] steht mindestens ein Interlock an
[19]	CPU_STATUS_RECEIVING_SYSPARAMETERS_RAM	R	Wenn [1] werden gerade Parameter vom PC empfangen
[18]	CPU_STATUS_MODULES_VERIFIED	R	Wird [1] sofern der FSP243 erfolgreich die gefundenen mit den gewünschten Modulen verglichen hat.
[17]	CPU_STATUS_PARAMETERS_VALID	R	Wenn [1] sind die in der MFU geladenen Parameter gültig
[16]	CPU_STATUS_LOADING_INTERNAL_PARAMETERS	R	Wenn [1] werden gerade die MFU internen Parameter geladen
[15]	CPU_STATUS_WATCHDOG	R	Perm. Wechsel zwischen [1] und [0] als Herzschlag der CPU Funktionalität
[14]	CPU_STATUS_BOOTSEQUENZ_COMPLETED	R	[1] wenn die MFU die Bootsequenz nach dem PowerUp abgeschlossen hat
[13]	CPU_STATUS_VNC1L_NOT_PROGRAMMED	R	[1] wenn der VNC1L/VNC2 (USB Controller) nicht programmiert ist
[12]	CPU_STATUS_USB_DEVICE_PERMITTED	R	Nur wenn [1] dürfen USB-, bzw. bestimmte Menüzugriffe erfolgen. Wird [1] wenn ein "zugelassener" USB Speicher an die MFU

			angeschlossen wird
[11]	CPU_STATUS_USB_DEVICE_DETECTED	R	Wenn [1] wurde ein eingesteckter USB Speicher an der MFU Front entdeckt
[10]	CPU_STATUS_USING_INTERNAL_PARAMETERS	R	Wenn [1] werden die internen Systemparameter aus dem seriellen Flash benutzt. Wenn [0] wurden diese Parameter von extern flüchtig überschrieben.
[9]	CPU_STATUS_ERROR_OCCURED	R	Wenn [1] ist ein Software Fehler aufgetreten. Dieser wird sowohl im Logbuch als auch flüchtig im Speicher vermerkt. Dient i.d.R. nur zu Debug-Zwecken.
[8]	CPU_STATUS_WARNING_OCCURED	R	Wenn [1] ist eine Systemwarnung aufgetreten welche nicht zwangsläufig das Gerät abgeschaltet hat, den Benutzer jedoch darauf hinweist.
[7]	CPU_STATUS_RECORDING_SYSPARAMETERS	R/W	Beim Übergang von [0] nach [1] wird der Parametersektor im seriellen Flash der MFU gelöscht. Anschließend werden alle über den USB empfangenen USI Kommandos im seriellen Flash gespeichert. Beim Übergang von [1] nach [0] wird der Speichervorgang abgeschlossen.
[6]	CPU_STATUS_FETCHING_INTERLOCKS	R	Wenn [1] globale Info darüber, dass gerade Interlocks von der CPU getestet werden
[5]	MPU_STATUS_MFU_CAN_NOT_ TRANSFER_ANY_DATA_RIGHT_NOW	R	Wenn [1] können gerade KEINE Daten an die MFU gesendet, bzw. von dieser gelesen werden, weil diese z.B. einen USI (Re)scan oder USB-Speicherzugriffe durchführt.
[4]	CPU_STATUS_RESET_BUTTON_ACTIVE	R	Wenn [1] ist die RESET Taste an der MFU Vorderseite gedrückt. Im Standardbildschirm kann nun anstelle 1/50 in 1/200 (Standard) Schritten der Handsollwert geändert werden. Andere Graduierungen lassen sich mittels [FSP249_SW_Local_Setvalue_Scaling_Factor] (Seite 31) einstellen.
[3]	CPU_STATUS_STDSCREEN_ACTIVE	R	Der Standardbildschirm ist aktiv und wird angezeigt.
[2]	CPU_STATUS_PSU_IS_REMOTE	R	Wenn [1] ist die MFU im Remote Betrieb.
[1]	-	R	n.u.
[0]	CPU_STATUS_PSU_IS_ON	R	Das ACU System ist eingeschaltet (Regler ist freigegeben)

Name	**FSP243_SW_VerifyHWConfig_ModuleClasses**
Adresse	**0xF3/243$_D$/0x4633$_{ASCII}$**
Tiefe	dynamisch
I/O	schreiben
Reset	0x(siehe Beschreibung)$_H$

Über diesen FSP lässt sich die korrekte Hardwarekonfiguration des ACU-Systems verifizieren.

Dieses FSP beinhaltet dabei die Modul(-Sub-)klassen der an den USIs erwarteten Module. Der NiosII der MFU liest dieses FSP und vergleicht dessen Inhalt mit den Modul(-Sub-)klassen der gefundenen Module. Besteht Konsistenz wird die Reglerfreigabe ermöglicht.

Die Datentiefe dieses FSP ist abhängig von der ACU Konfiguration und der Anzahl der an der MFU angebundenen Module.

Die in diesem FSP zu verarbeitenden Daten sind dabei wie folgt organisiert:

```
STX PID PID GW MA FSP FSP { [Head Head][ModuleClass
ModulClass][ModuleSubClass ModulSubClass] } PP PP ETX
```

mit

`[Head]`	oberes Nibble USI Nummer, unteres Nibbel Modul Nummer Bsp.: 0x12$_H$/0x31,0x32$_{ASCII}$ für USI 1, Modul 2
`[ModuleClass]`	2 Nibble Hex (2 Byte ASCII) Modulklasse des in [Head] beschriebenen Moduls
`[ModuleSubClass]`	2 Nibble Hex (2 Byte ASCII) Modul-Sub-Klasse des in [Head] beschriebenen Moduls
`{}`	stetige Wiederholung obiger Beschreibung für alle Module

Wird dieses FSP gefüllt, wird sofort die gefundene Modulkonfiguration mit der in diesem FSP gewünschten verglichen. Ist dieser Vergleich erfolgreich, wird ‚CPU_STATUS_MODULES_VERIFIED' im FSP242 gesetzt.

Wird die MFU ohne Module betrieben lässt sich das Flag ‚CPU_STATUS_MODULES_VERIFIED' setzen, indem ein Schreibkommando an FSP243 gesendet wird, bei dem alle Parameter (Head, ModulClass, ModulSubClass) auf null (0x30) stehen.

STX	PID	GW	MA	FSP	Head	ModuleClass	ModuleSubClass	PP	ETX
0x02	0x57 0x52	0x30	0x30	0x46 0x33	0x30 0x30	0x30 0x30	0x30 0x30	0x30 0x30	0x03

Name	**FSP244_SW_ChangeUSIBitrate_ChangeUSIMode**
Adresse	**0xF4/244$_D$/0x4634$_{ASCII}$**
Tiefe	2 Byte / 16 Bit
I/O	schreiben
Reset	0x(siehe Beschreibung)$_H$

Über diesen FSP kann seitens des PC die Bitrate sowie der USI Modus einzelner USI konfiguriert werden.

Die Anforderung ist dabei identisch mit einer Anforderung an das Standard FSP012 zur Konfiguration der USI. Schreibt der PC direkt an einen Modul FSP012, erfolgt keine Umschaltung der USI innerhalb der MFU, da die MFU lediglich als Gateway funktioniert und die Daten zwischen PC und Modulen nicht analysiert.

Daher erfolgt die Bitratenumschaltung der USI über diesen Software FSP.

Mit diesem FSP wird die Bitraten/USI Mode Umschaltung in der MFU vorgenommen und von dieser mit dem/mit den angeschlossenen Modul(en) verhandelt, damit auch diese die gewünschte Bitrate/den gewünschten USI Mode entsprechend der Anforderung einstellen.

WICHITG: Ist eine USI in HighSpeed, kann deren Bitrate NICHT direkt in HighSpeed geändert werden. Es muss zuvor HighSpeed an dieser USI dekaktiviert werden. Bei der HighSpeed-Deaktivierungs-Anforderung kann aber eine Wunschbitrate für die USI-Standardkommunikation mitgeteilt werden. (Bsp.: mit `Data Data` = 0x30 0x38 wird HighSpeed abgeschaltet und das USI wechselt sofort auf 10MBit). Im Anschluss wird ggf. in einer weiteren Anforderung an diesen FSP die neue Bitrate/der neue USI Mode eingestellt.

Sollen an einem USI mit mehreren Modulen alle Module zeitgleich auf eine neue Bitrate geschaltet werden, wird für den Parameter `Mod` = 0 (0x30) gewählt.

Sollen alle Module an allen USIs zeitgleich auf eine neue Bitrate geschaltet werden, wird für den Parameter `USI` = 0 (0x30) gewählt.

```
STX PID PID GW MA FSP FSP USI Mod Data Data PP PP ETX
```

`STX`	- StartOfText	(0x02)
`PID PID`	- Schreibanforderung	(0x5752)
`GW`	- Gateway Adresse	(0x30 – da Gateway die MFU ist)
`MA`	- Modul Adresse	(0x30 – da Modul die MFU ist)
`FSP FSP`	- FSP Nummer	(0x46, 0x34)
`USI`	- USI Nummer an der Bitrate/Modus geändert werden soll (0x30, 0x31...0x42)	
`Mod`	- Modulnummer an dem Bitrate/Modus geändert werden soll (0x30, 0x31...0x38)	
`Data Data`	- Bitrate/Modus (siehe Standard FSP012)	
`PP PP`	- Prüfsumme	
`ETX`	- EndOfText	(0x03)

Mit `Data` für Standrad USI:
07 `[0x30 0x37]` = 115,2kBit
06 `[0x30 0x36]` = 1Mbit
05 `[0x30 0x35]` = 2Mbit
04 `[0x30 0x34]` = 5Mbit
03 `[0x30 0x33]` = 10Mbit
02 `[0x30 0x32]` = 16,6Mbit
01 `[0x30 0x31]` = 20Mbit
00 `[0x30 0x30]` = 25MBit

Soll USI HighSpeed benutzt werden, muss zusätzlich das MSB gesetzt werden:
87 `[0x38 0x37]` = 115,2kBit HS
86 `[0x38 0x37]` = 1MBit HS
(...)

Example: switch USI 1 Module 1 to 115,2kBit, no HighSpeed

```
STX W  R  0  0  F  4  1  1  0  7  0  7  ETX
02  57 52 30 30 46 34 31 31 30 37 30 37 03
```

Example: switch on HighSpeed for USI 1 Module 1 again and stay at 115.2kBit

STX	W	R	0	0	F	4	1	1	8	7	0	F	ETX
02	57	52	30	30	46	34	31	31	38	37	30	46	03

Example: switch on HighSpeed for USI 1 Module 1 again and switch directly to 2Mbit

STX	W	R	0	0	F	4	1	1	8	5	0	D	<3>
02	57	52	30	30	46	34	31	31	38	35	30	44	03

Name	**FSP245_SW_intScopeDataStream**
Adresse	**0xF5/245$_D$/0x4635$_{ASCII}$**
Tiefe	12008 Byte / 96064 Bit (bis MFU FW 7.4.x) 6008 Byte / 48064 Bit (ab MFU FW 7.5.0)
I/O	lesen
Reset	0x(siehe Beschreibung)$_H$

Dieser FSP sendet Scope Kanaldaten des internen Scopes mit

8 Kanälen (bis MFU FW 7.4.x) mit je 500 Werten und 12 Bit Tiefe
→ 6000 Byte$_H$, bzw. 12000 Bytes$_A$ + einbettende STX ... PP PP ETX usw.

4 Kanälen (ab MFU FW 7.5.0) mit 500 Werten und 12 Bit Tiefe
→ 3000 Byte$_H$, bzw. 6000 Bytes$_A$ + einbettende STX ... PP PP ETX usw.

Bei der Ausgabe dieser Daten wird der Triggerpunkt und -offset berücksichtigt.

D.h. die Daten werden nicht beginnend vom RAM Speicher 0, sondern beginnend vom Triggeroffset ausgegeben. An der höchsten RAM-Stelle (500) wird mit der RAM Stelle 0 fortgefahren und die Ausgabe endet bei Triggeroffset – 1.

Sollen die Daten beginnend vom RAM Speicher 0 und nicht vom Triggeroffset ausgegeben werden bitte den FSP231_SW_intScopeDataStreamReadOut benutzen.

FSP245 kann nur gelesen werden. Schreibanforderungen bekommen eine NACK Antwort mit Fehlercode.

Bis MFU FW 7.4.x gilt folgendes:

Ausgabestring nach der Leseanforderung:

```
STX GW MA -FSP-  -CH8(12Bit) CH7(12Bit) CH7(12Bit) CH5(12Bit)
CH4(12Bit) CH3(12Bit) CH2(12Bit) CH1(12Bit)- -PP-   ETX

02  30 30 45 37   dd dd dd   dd dd dd   dd dd dd   dd dd dd
dd dd dd   dd dd dd   dd dd dd   dd dd dd    pp pp  03
```

CHx - Daten des Kanals

Insgesamt werden 500 x **8** Kanäle übertragen. D.h. der gelb markierte Bereich wird also insgesamt 500-mal übertragen. Beginnend mit dem Wert 1 für Kanal 8 bis 1, Wert 2 für Kanal 8 bis 1 usw. beginnend beim Triggerpunkt. Es bilden also immer 8 x 12Bit = 96Bit → 12 Bytes einen Datensatz. Die Prüfsumme PP folgt nach Wert 500 für Kanal 8 bis 1 über die gesamten Daten.

Ab MFU FW 7.5.0 gilt folgendes:

Ausgabestring nach der Leseanforderung:

```
STX GW MA -FSP-  -CH4(12Bit) CH3(12Bit) CH2(12Bit) CH1(12Bit)- -PP-
ETX

02  30 30 45 37   dd dd dd   dd dd dd   dd dd dd   dd dd dd    pp pp
03
```

CHx - Daten des Kanals

Insgesamt werden 500 x **4** Kanäle übertragen. D.h. der gelb markierte Bereich wird also insgesamt 500-mal übertragen. Beginnend mit dem Wert 1 für Kanal 4 bis 1, Wert 2 für Kanal 4 bis 1 usw. beginnend beim Triggerpunkt. Es bilden also immer 4 x 12Bit = 48Bit → 6 Bytes einen Datensatz. Die Prüfsumme PP folgt nach Wert 500 für Kanal 4 bis 1 über die gesamten Daten.

Name	**FSP246_SW_Recorded_Supplies**
Adresse	**0xF6/246$_D$/0x4636$_{ASCII}$**
Tiefe	10800 Byte
I/O	lesen
Reset	0x(siehe Beschreibung)$_H$

Dieser FSP liefert die in der MFU überwachten Betriebsspannungen der letzten 24 Stunden. Die Spannungen werden einmal je Minute gespeichert. Die Sortierung ist dabei wie folgt: 1V2, 2V5, 3V3, 5V0, p12V0, n12V0. Der Speicher der Betriebsspannungen ist flüchtig, d.h. nach dem Aus- und Wiedereinschalten sind alle Werte zunächst 0.

Je Spannung wird durch 12 Bit Daten + Vorzeichen repräsentiert. Bei der Ausgabe werden je Spannung 16 Bit gesendet. Das MSB ist dabei je Spannung das Vorzeichen, danach folgen die 12 Bit Spannung. Die restlichen 3 Bit je Spannungswert bleiben immer 0.

FSP246 kann nur gelesen werden. Schreibanforderungen bekommen eine NACK Antwort mit Fehlercode.

Ausgabestring nach der Leseanforderung:

```
STX GW MA -FSP-  1V2(16Bit)   2V5(16Bit)   3V3(16Bit)   5V0(16Bit)
12V0(16Bit)  -12V0(16Bit) (…) PP PP ETX

02  30 30 46 36   dd dd dd dd  dd dd dd dd  dd dd dd dd  dd dd dd dd
dd dd dd dd  dd dd dd dd             03
```

dd dd dd dd$_H$ entspricht dabei vsss ssss ssss sxxx$_B$ mit

v = Vorzeichen
s = Spannungswert
x = don't care, immer ‚0'

Insgesamt werden 24x60=1440 Werte je Spannung übertragen. D.h. der gelb markierte Bereich wird insgesamt 1440-mal übertragen. Beginnend mit dem Wert für 1,2 Volt. Die Prüfsumme PP folgt nach dem 1440-sten Wert für -15,0 Volt über die gesamten Daten.

1440 Spannungen * 6 Einträge mit je 16 Bit 17280 Byte Hex, bzw. 34560 Byte ASCII an Daten.

Die Bitwertigkeiten sind wie folgt:

bei 1V2, 2V5, 3V3 und 5V0 werden die gemessenen Werte direkt mit dem LSB Wert (SUP_LSB_Size = 0.002441) multipliziert und ergeben so die Spannung. Bei -12V0 und 12V0 erfolgt ebenfalls zuerst die LSB Wert Multiplikation und anschließend die Multiplikation mit dem Faktor 11, der dem vorgeschalteten Spannungsteiler entspricht. Die tatsächlich gemessenen Spannungen liegen nämlich bei 1/11 der Originalspannung.

Name	**FSP247_SW_Recorded_Temperatures**
Adresse	**0xF7/247$_D$/0x4637$_{ASCII}$**
Tiefe	8640 Byte
I/O	lesen
Reset	0x(siehe Beschreibung)$_H$

Dieser FSP liefert die in der MFU überwachten Modultemperaturen der letzten 24 Stunden. Die Temperaturen werden einmal je Minute gespeichert. Die Sortierung ist dabei wie folgt: FPGA, Längsregler, Modulmitte. Der Speicher der Temperaturenaufzeichnung ist flüchtig, d.h. nach dem Aus- und Wiedereinschalten sind alle Werte zunächst 0.

Je Temperatur steht ein 8 Bit, also 1 Byte Speicher zur Verfügung. Alle 3 Temperaturen belegen also pro Eintrag 3 Byte Hex, bzw. 6 Byte ASCII.

FSP247 kann nur gelesen werden. Schreibanforderungen bekommen eine NACK Antwort mit Fehlercode.

Ausgabestring nach der Leseanforderung:

```
STX GW MA -FSP-  T1(8Bit) T2(8Bit) T3(8Bit) (…) PP PP ETX
02  30 30 46 37   dd dd    dd dd    dd dd              03
```

Insgesamt werden 24x60=1440 Werte je Temperatur übertragen. D.h. der gelb markierte Bereich wird insgesamt 1440-mal übertragen. Beginnend mit dem Wert T1 für das FPGA. Die Prüfsumme PP folgt nach dem 1440-sten Wert für T3 über die gesamten Daten.

1440 Übertragungen mit je 3 Byte Hex sind 4320 Byte Hex, bzw. 8640 Byte ASCII an Daten.

Die Bitwertigkeiten sind wie folgt:

Aktuelle Temperatur	Gemessene Temperatur	Binär Hexadzimal
+130.00°C	+127°C	0111 1111
+127.00°C	+127°C	0111 1111
+126.50°C	+126°C	0111 1110
+25.25°C	+25°C	0001 1001
+0.50°C	0°C	0000 0000
+0.25°C	0°C	0000 0000
0.00°C	0°C	0000 0000
-0.25°C	-1°C	1111 1111
-0.50°C	-1°C	1111 1111
-0.75°C	-1°C	1111 1111
-1.00°C	-1°C	1111 1111
-25.00°C	-25°C	1110 0111
-25.25°C	-26°C	1110 0110
-54.75°C	-55°C	1100 1001
-55.00°C	-55°C	1100 1001
-65.00°C	-65°C	1011 1111

Name	**FSP248_SW_ReadExtRAMData**
Adresse	**$0xF8_H/248_D/0x4638_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Gibt eine bestimmte, frei wählbare Anzahl von RAM Einträgen eines ExtRAMExtention Moduls zurück.

Dazu müssen zunächst Daten gesammelt worden sein.

Die Daten können nur 1x ausgelesen werden. Danach muss ein neuer Speicherzyklus durchlaufen werden. Vor dem Starten eines neuen Speicherzyklus müssen ALLE Daten ausgelesen worden sein.

Zum Auslesen der Daten muss der FSP ggf. zunächst beschrieben werden mit:

```
STX PID PID GW MA FSP FSP [Quantity][Quantity][Quantity][Quantity] PP
PP ETX
```

Mit:

`PID PID` WR
`[Quantity]` Anzahl von Werten (1..65536)

Der Resetwert von `[Quantity]` ist 0x0000, dadurch werden alle im ExtRAMmodul befindlichen Werte zurück geliefert.

Sofern `[Quantity]` != 0x0000 gesetzt wurde, lässt `[Quantity]` sich nach jedem ausgelesenen Datenblock beliebig ändern und auch wieder auf 0x0000 setzen um die restlichen Daten in einem Aufruf zu lesen.

Wird der FSP anschließend gelesen, werden mit jeder Leseaufforderung [Quantity] Datensätze des ExtRAMExtention Moduls gesendet. Ist `[Quantity]` = 0 werden alle Daten ohne Unterbrechung gesendet:

```
STX GW MA FSP FSP
[Byte3 Byte2 Byte1 Byte0] ← Datensatz n
[Byte3 Byte2 Byte1 Byte0] ← Datensatz n-1
[..]
[Byte3 Byte2 Byte1 Byte0] ← Datensatz n-n
PP PP ETX
```

Name	**FSP249_SW_Local_Setvalue_Scaling_Factor**
Adresse	**$0xF9_H/249_D/0x4639_{ASCII}$**
Tiefe	2 Byte /16 Bit
I/O	lesen / schreiben
Reset	$0x0002_H$

Definiert das Teilerverhältnis der Lokalen, manuellen Sollwertvorgabe.

Die Standardschrittweite bei der Sollwertvorgabe beträgt 1/50 des Nennwerts. Bei Betätigen der RESET-Taste im Local-Modus wird diese Schrittweite weiter heruntergeteilt.

:2, :4(Default), :8, :16, :32, :64, :128, ... (usw.)

Weil bei der Berechnung des Teilerfaktors dieser einfach geschoben wird, muss 'setvalue_stepwidth' die Anzahl der Bit-Schübe beinhalten.

Der Standardwert ist hier 2, dadurch wird 1/50 zu 1/200.

```
STX PID PID GW MA FSP FSP [Shift][Shift] PP PP ETX
```

Mit:

`[Shift]` Anzahl von Bitschüben

Name	**FSP250_SW_NIOS_Software_ Version**
Adresse	**$0xFA_H/250_D/0x4641_{ASCII}$**
Tiefe	dynamisch
I/O	lesen
Reset	$0x(\text{siehe Beschreibung})_H$

Dieser SW FSP beinhaltet die NiosII SW Version für Remote Updates.

Er kann nur gelesen werden.

Nachfolgendes Beispiel zeigt den NiosII Software-Stand 007.00004 an.

□	0	0	F	A	0	0	7	.	0	0	0	0	4	2	D	□
02	30	30	46	41	30	30	37	2E	30	30	30	30	34	32	44	03

Name	**FSP251_SW_compressed_PCA_configuration_file**
Adresse	**$0xFB_H/252_D/0x4642_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Dieser SW FSP beinhaltet komprimierte PCA Konfigurationsdatei.

Um eine Datei zu speichern muss zunächst ein Schreibkommando auf den FSP erfolgen, in dem die Größe der anschließend zu übertragenden *.xpc7-Konfigurationsdatei mitgeteilt wird.

```
STX PID PID GW MA FSP FSP –[Length (8 Byte)]-       PP PP ETX

STX  W   R   0  0   F   B  Dateigroesse in Byte     PP PP ETX
```

mit

`[Length]` ist 8 Byte ASCII lang, sollen z.B. $8.977_D = 2311_H$ Byte übertragen werden ist
`Length = 0x30 0x30 0x30 0x30 0x32 0x33 0x31 0x31`

Im Falle von ACK schaltet die MFU automatisch auf einen Empfangsmodus um und erwartet nachfolgend die zuvor angekündigte Anzahl von Bytes. D.h. es werden nun solange Daten der komprimierten *.xpc7 Datei an die MFU gesendet, bis alle Daten im seriellen Flash gespeichert sind.

Die Daten werden dabei NICHT als ASCII-Zeichen, sondern wie sie vorliegen, als reine Rohdaten gesendet.

Der Empfang der Daten wird abschließend mit einem weiteren ACK, bzw. NACK quittiert.

Soll die Datei aus dem seriellen Flash gelesen werden, muss lediglich das FSP gelesen werden.

Die dabei empfangenen Daten werden in einen USI-Header, eine Prüfsumme und ein ETX eingebettet.

Die Daten an sich sind aber wie beim Senden auch, die reinen Rohdaten und NICHT ASCII kodiert.

Name	**FSP252_SW_UpdateMFU_CFI_SoftwareViaRemote**
Adresse	**$0xFC_H/252_D/0x4643_{ASCII}$**
Tiefe	Dynamisch
I/O	lesen / schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Das CFI beinhaltet 2 Software-Versionen, die 'Factory-SW' und die 'Application-SW' und zusätzlich den Boot Copier

--------- SCHREIBEN ---------

Via FSP252 können beide Softwaren und der Boot Copier ersetzt werden.

Per Default wird aber i.d.R. nur die Application-SW ersetzt.

Dazu muss das FSP252 zunächst ein Init-Kommando erhalten:

```
STX PID PID GW MA FSP FSP -Data-                    PP PP ETX
```

(Update Boot Copier)

```
STX  W  R   0  0  F   C   UBC+Dateigroesse in Byte  PP PP ETX
```

(Update Application SW)

```
STX  W  R   0  0  F   C   UAS+Dateigroesse in Byte  PP PP ETX
```

(Update Factory SW)

```
STX  W  R   0  0  F   C   UFS+Dateigroesse in Byte  PP PP ETX
```

Die [`Dateigroesse in Byte`] muss immer 8 Zeichen lang sein. Die Angabe der Dateigröße erfolgt in Hexadezimal. Führende Stellen sind 0.

Nachfolgend soll als Beispiel das Application-Image ersetzt werden. Die dazu nötige *.s19-Datei hat eine Größe von 959232_D = $EA300_H$ Bytes.

□	W	R	0	0	F	C	U	A	S	0	0	0	E	A	3	0	0	4	0	□
02	57	52	30	30	46	43	55	41	53	30	30	30	45	41	33	30	30	34	30	03

Die Prüfsumme ist hierbei über den gesamten `-Data-` Bereich (`UAS000EA300`) zu bilden.

Darauf antwortet FSP252 mit einem ACK, bzw. im Fehlerfall einem NACK.

Im Falle von ACK schaltet die MFU automatisch auf einen Empfangsmodus um und erwartet nachfolgend die zuvor angekündigte Anzahl von Bytes. D.h. es werden nun solange Daten der *.s19-Datei an die MFU gesendet, bis alle Daten im CFI gespeichert sind. Die Daten werden dabei NICHT als ASCII-Zeichen, sondern wie sie vorliegen, als reine Rohdaten gesendet.

Der Empf. der *.s19-Datei wird mit ACK, bzw. NACK quittiert.

--------- LESEN ---------

Sollen Daten aus dem CFI gelesen werden, muss dies zuvor per Schreibzugriff definiert welche Daten genau wie viele Bytes davon gelesen werden sollen.

(Read Boot Copier)

```
STX  W  R   0  0  F   C   RBC+Dateigroesse in Byte  PP PP ETX
```

(Read Application SW)

```
STX  W  R   0  0  F   C   RAS+Dateigroesse in Byte  PP PP ETX
```

(Read Factory SW)

```
STX  W  R   0  0  F   C   RFS+Dateigroesse in Byte  PP PP ETX
```

Anschließend erfolgt eine Leseaufforderung. Diese liefert dann die gewünschte Anzahl an Bytes.

Bsp.: 390000_D = $5F370_H$ Byte der Application SW lesen:

□	W	R	0	0	F	C	R	A	S	0	0	0	5	F	3	7	0	3	7	□
02	57	52	30	30	46	43	52	41	53	30	30	30	35	46	33	37	30	33	37	03

Die Prüfsumme ist hierbei über den gesamten `–Data–` Bereich (`RAS0005F370`) zu bilden.

Anschließend erfolgt ein Lesekommando. Die dabei empfangenen Daten werden in einen USI-Header, eine Prüfsumme und ein ETX eingebettet.

Die Daten an sich sind aber wie beim Senden auch, die reinen Rohdaten und NICHT ASCII kodiert.

--------- LÖSCHEN ---------

Das CFI lässt sich auch nur löschen (vollständig):

`STX W R 0 0 F C EBC+Dateigroesse in Byte PP PP ETX`

Die Daten für [`Dateigroesse in Byte`] sind dabei egal.

Name	**FSP253_SW_UpdateMFU_EPCS_FirmwareViaRemote**
Adresse	**0xFD$_H$/253$_D$/0x4644$_{ASCII}$**
Tiefe	dynamisch
I/O	lesen / schreiben
Reset	0x(siehe Beschreibung)$_H$

Das EPCS beinhaltet 2 Firmware-Versionen, das 'Factory-Image' und das 'Application-Image'.

Via FSP253 können beide Images ersetzt werden.

Per default wird aber i.d.R. nur das Application-Image ersetzt.

Dazu muss das FSP253 zunächst ein Init-Kommando erhalten:

```
STX PID PID GW MA FSP FSP -Data-                    PP PP ETX
```

(Update Application Image)

```
STX W   R   0  0  F   D   UAI+Dateigroesse in Byte  PP PP ETX
```

Beim Factory Image muss das Init-Kommando entsprechend so lauten:

(Update Factory Image)

```
STX  W  R   0  0  F   D   UFI+Dateigroesse in Byte  PP PP ETX
```

Die `Dateigroesse in Byte` muss immer 8 Zeichen lang sein. Die Angabe der Dateigröße erfolgt in Hexadezimal. Führende Stellen sind 0.

Nachfolgend soll als Beispiel das Application-Image ersetzt werden. Die dazu nötige *.rbf-Datei hat eine Größe von 988394$_D$ = F14EA$_H$ Bytes.

¤	W	R	0	0	F	D	U	A	I	0	0	0	F	1	4	E	A	2	A	¤
02	57	52	30	30	46	44	55	41	49	30	30	30	46	31	34	45	41	32	41	03

Die Prüfsumme ist hierbei über den gesamten `-Data-` Bereich (`UAI000F14EA`) zu bilden.

Daraufhin antwortet FSP253 mit einem ACK, bzw. im Fehlerfall einem NACK.

Im Falle von ACK schaltet die MFU automatisch auf einen Empfangsmodus um und erwartet nachfolgend die zuvor angekündigte Anzahl von Bytes. D.h. es werden nun solange Daten der *rbf-Datei an die MFU gesendet, bis alle Daten im EPCS gespeichert sind. Die Daten werden dabei NICHT als ASCII-Zeichen, sondern wie sie vorliegen, als reine Rohdaten gesendet

Der Empfang der *.rbf Daten wird mit ACK, bzw. NACK quittiert.

Sollen Daten aus dem EPCS gelesen werden, muss zuvor per Schreibzugriff definiert werden ob das Application oder das Factory Image gelesen werden soll und wie viele Bytes davon gelesen werden sollen.

(Read Appilcation Image)

```
STX  W  R   0  0  F   D   RAI+Dateigroesse in Byte  PP PP ETX
```

(Read Factory Image)

```
STX  W  R   0  0  F   D   RFI+Dateigroesse in Byte  PP PP ETX
```

Bsp.: 988394$_D$ = F14EA$_H$ Byte der Application FW lesen

¤	W	R	0	0	F	D	R	A	I	0	0	0	F	1	4	E	A	2	D	¤
02	57	52	30	30	46	44	52	41	49	30	30	30	46	31	34	45	41	32	44	03

Die Prüfsumme ist hierbei über den gesamten `-Data-` Bereich (`RAI000F14EA`) zu bilden.

Anschließend erfolgt ein Lesekommando. Die dabei empfangenen Daten werden in einen USI-Header, eine Prüfsumme und ein ETX eingebettet.

Die Daten an sich sind aber wie beim Senden auch, die reinen Rohdaten und NICHT ASCII kodiert.

Name	**FSP254_SW_Parameter_Information_String**
Adresse	**$0xFE_H/254_D/0x4645_{ASCII}$**
Tiefe	Dynamisch, max. 256 Zeichen
I/O	lesen / schreiben
Reset	0x(siehe Beschreibung)$_H$

Parameterinformationen in ASCII Klartext.

```
STX PID PID GW MA FSP FSP [Text] PP PP ETX
```

Mit:

`[Text]` max. 256 Zeichen Textinformation über die in der MFU hinterlegten Parameter

Name	**FSP255_SW_Flash_VNC2**
Adresse	**$0xFF_H/255_D/0x4646_{ASCII}$**
Tiefe	65536 Byte / 524288 Bit
I/O	schreiben
Reset	$0x(\text{siehe Beschreibung})_H$

Der interne FSP255 ist ein CPU Software FSP und wird zum programmieren des Vinculum VNC2 USB Master uController in der MFU benutzt.

FSP255 kann nur geschrieben werden. Leseanforderungen bekommen eine NACK Antwort mit Fehlercode.

Bis MFU Software 002.00002

Zum programmieren des Vinculum VNC2 wird eine reguläre USI Schreibanforderung an dieses FSP gesendet. Die nach dem Header folgenden Daten werden von der MFU bei diesem FSP ohnehin verworfen, d.h. es brauchen weder Daten noch eine Prüfsumme vor dem ETX gesendet werden.

```
STX PID PID GW MA FSP FSP ETX
```

STX	-	StartOfText	(0x02)
PID PID	-	Schreibanforderung	(0x5752)
GW	-	Gateway Adresse	(0x30 – da Gateway die MFU ist)
MA	-	Modul Adresse	(0x30 – da Modul die MFU ist)
FSP FSP	-	FSP Nummer	(0x46, 0x46)
[Daten..Prüfsumme]	- nicht notwendig, kann entfallen		
ETX	-	EndOfText	(0x03)

Wichtig: Sofern die Programmieranforderung von der MFU verstanden wurde, wird KEIN ACK gesendet. Wird diese hingegen nicht verstanden erfolgt das senden eines NACK.

Im Erfolgsfall wechselt die Anzeige des LCD und zeigt „Waiting for flash ROM data“.

Nun wird das ROM File gesendet. Wichtig ist hierbei, dass die Übertragung in 'Hex' erfolgt.

Erst nach dem erfolgreichen senden und programmieren des ROM Files wird von der MFU ein ACK, im Fehlerfall ein weiteres NCK gesendet.

Ab NIOS Software 002.00003

Zum programmieren des Vinculum VNC2 wird eine reguläre USI Schreibanforderung an dieses FSP gesendet. Nach dem Header wird der MFU die Länge der folgenden Programmierdatei mitgeteilt. Dies ist notwendig, weil die MFU auf dem TFT eine Fortschrittsanzeige generiert und der VNC2 nur vollständig geschriebene Pages akzeptiert. Ist die Programmierdatei zu Ende aber die letzte Page des VNC2 noch nicht vollständig programmiert schlägt die gesamte Programmierung fehl. Die MFU generiert aus der Information der Programmierdatenlänge und den evtl. abschließend fehlenden Daten eine vollständige letzte Page für den VNC2 automatisch. Dazu benötigt sie aber die Länge der Programmierdaten. Bis V002.00002 war die Länge per #define festgelegt, was das Programmieren des VNC2 mit neueren Softwareständen deren Länge davon abweicht nicht möglich machte.

```
STX PID PID GW MA FSP FSP Length Lenght Length Length PP PP ETX
```

STX	-	StartOfText	(0x02)
PID PID	-	Schreibanforderung	(0x5752)
GW	-	Gateway Adresse	(0x30 – da Gateway die MFU ist)
MA	-	Modul Adresse	(0x30 – da Modul die MFU ist)
FSP FSP	-	FSP Nummer	(0x46, 0x46)
Length	-	Länge der nachfolgenden Programmierdatei	(0x30, 0x30, 0x30, 0x33, 0x32, 0x30, 0x39, 0x30)
PP PP	-	Prüfsumme über Length	
ETX	-	EndOfText	(0x03)

Sofern die Programmieranforderung von der MFU verstanden wurde, wird EIN ACK gesendet die Anzeige des TFT zeigt „Waiting for flash ROM data“.

Wird diese hingegen nicht verstanden erfolgt das senden eines NACK.

Nun wird das ROM File gesendet. Wichtig ist hierbei, dass die Übertragung in 'Hex' erfolgt.

Nach dem erfolgreichen senden und programmieren des ROM Files wird von der MFU ein weiteres ACK, im Fehlerfall ein NACK gesendet.