Anhang F (Datagramm Kommunikation)

Allgemein

Zweck der Datagramm Kommunikation ist eine im Gegensatz zur TCP/IP Datenübertragung performantere Bereitstellung von Trace-Daten. Die Daten werden dabei in einer selbstbeschreibenden Form bereitgestellt. Damit soll auch Hostapplikationen ohne SCPI Datenschnittstelle die Auswertung der Daten ermöglicht werden. Das dafür notwendige Protokoll wird in diesem Abschnitt beschrieben. Die verschiedenen Datenarten werden über Tags identifiziert und mit Längeninformation versehen. Somit können Hostapplikationen die entsprechenden Daten ihres Interesses herausfiltern und verarbeiten und müssen nicht notwendigerweise das gesamte Protokoll implementierten. Siehe auch Anhang C “Beispielprogramm UDP“.

Adressierung

Für die Verteilung der Daten existiert das Konzept "**UdpPath**". Solch ein UdpPath beinhaltet eine IP-Adresse, eine bestimmte Port Nummer und Konfigurationsdaten. UdpPath ist somit nicht gleichzusetzen mit Host, da über die IP-Adresse auch mehrere Hosts gleichzeitig ansprechbar sind (über Broadcast / Multicast Adressiering) und andererseits ein Host auch mehrere UdpPath's bedienen kann (verschiedene Port Nummern mit unterschiedlichen Konfigurationen).

Konfiguration

Die Konfiguration der verschiedenen UdpPath Einträge beinhaltet erstens die Art der Daten. Über unterschiedliche **Tags** können die verschiedenen Trace-Typen konfiguriert werden. Über unterschiedliche **Flags** können zusätzlich die gewünschten Trace Daten genauer spezifiziert werden.

Protokoll

Jedes Datagramm (Udp-Paket) besitzt einen **Header**, der besagt, daß es sich zweifelsfrei um ein Datum dieses Protokolls handelt. Danach folgen ein oder mehrere als **GenericAttribute** bezeichnete Dateneinheiten, die durch verschiedene Tags unterschieden werden können. Zur Protokoll-Spezifikation wird ein C-ähnliche Notation verwendet. Normalerweise werden Daten in der Network-Byte-Order, also in der Big-Endian-Order übertragen. Dies ist im besonderen für Hostapplikationen auf PC (Intel) Basis von Bedeutung, da in diesem Fall die Daten erst in das Little-Endian-Format umgewandelt werden müssen. Jedoch können die eigentlichen Nutzdaten auch in der Little-Endiean-Order übertragen werden (siehe später Beschreibung Remote Befehle).

EB200Datagram {

    EB200Header

    GenericAttribute_1

    GenericAttribute_2

    ...

    GenericAttribute_n

}

Header

Der Header leitet jedes EB200 Datagramm ein.

EB200Header {

```
    unsigned long magic_number;              /* constant: 0x000EB200 (auch bei ESMB) */
    unsigned short minor_version_number;     /* 0x26 für diese Version */
    unsigned short major_version_number;     /* 0x02 für diese Version */
    unsigned short sequence_number;          /* inkrementiert um eins */
    char[6] reserved;
}
```

Beschreibung des EB200 Headers:

- **magic_number**
  Dies ist ein konstanter Wert und wird sich nie ändern.
- **minor_version_number** und **major_version_number**
  Bei Hinzufügen von Tags oder Änderungen von Tags in einer aufwärtskompatiblen Form wird die minor_version_number inkrementiert. Bei nichtkompatiblen Änderungen von Tags oder generellen strukturellen Änderungen wird die major_version_number inkrementiert (letzteres wird, wenn überhaupt, sehr selten der Fall sein).
- **sequence_number**
  Diese startet bei einem bestimmten Wert und wird pro UdpPath mit jedem neuen Paket um eins erhöht. Beim höchsten Wert angekommen, passiert ein Wrap-Around.
- **reserved**
  Dieses Element wird für etwaige zukünftige Erweiterungen freigelassen.

Ab *minor_version_number 0x24* wird beim DSCAN-Datagramm im *OptionalHeader* das zusätzliche Flag *newStepScheme* übertragen. Dieses Flag zeigt, ob das alte oder das neue Kanalraster gewählt wurde.

Ab *minor_version_number 0x26* wird bei AUDIO ein Demodulationsstring mitgeschickt.

GenericAttribute

Hier wird einmal die Allgemeine Struktur eines jeden folgenden Datenelement (als GenericAttribute) beschrieben. Alle Datentypen (also alle Trace-Daten gekennzeichnet durch die unterschiedlichen Tags) besitzen den gleichen Aufbau.

GenericAttribute {

unsigned short tag;

unsigned short length;

char data[length];

}

- **tag**
  Bestimmt den eigentlichen Inhalt dieses GenericAttributes.
- **length**
  kennzeichnet die Länge dieses GenericAttributes in Bytes ausschließlich der Elemente "tag" und "length".
- **data**
  hier kommen die eigentlichen Nutzdaten zu liegen. Die Länge dieses Datenbereichs in Bytes wird durch das vorangegangene Datenelement "length" bestimmt.

Es folgt nun die Beschreibung der verschiedenen Tags.

Für die zum Zeitpunkt definierten Tags (FSCAN, MSCAN, DSCAN, AUDIO, IFPAN, FASTLEVCW, LIST und CW) existiert eine gemeinsame Grundstruktur, die hier vorab beschrieben werden soll. Es handelt sich um jene Struktur, die im Element "data" des GenericAttributes zu liegen kommt.

Symbolischer TAG Name	*Numerischer TAG Wert (dezimal)*
FSCAN	101
MSCAN	201
DSCAN	301
AUDIO	401
IFPAN	501
FASTLEVCW	601
LIST	701
CW	801

Tabelle 1: Beschreibung der TAGs

TraceAttribute

Die allgemeine Struktur aller bis jetzt definierten Trace-Daten wird hier beschrieben.

TraceAttribute {

```
  short number_of_trace_items;          /* Anzahl der Werte pro Datentyp in PeriodicTraceData */
  char reserved;
  unsigned char optional_header_length;/* Größe des Optional Headers in Bytes */
  unsigned long selectorFlags;          /* Genauere Spezifikation der Daten */

  OptionalHeader;                       /* Dieser wird beim entspr. Trace genauer beschrieben */

  PeriodicTraceData;                    /* die eigentlichen Trace Daten */
                                        /* je nach SelectorFlags bzw. Tag und OptionalHeader */
```

}

Es sollte immer der Wert in "optional_header_length" herangezogen werden, um zu den eigentlichen Trace Daten in PeriodicTraceData zu gelangen, um auch bei "minor_version_number"-Änderungen Aufwärtskompatibilität zu gewährleisten.

Die "selectorFlags" werden hier zwar allgemein beschrieben, machen aber bei den einzelnen Traces nur teilweise Sinn.

SelectorFlag	*Hexadezimal -Wert*	*Daten-Typ*	*Entsprechende Flags*	*Bemerkung*
LEVEL	0x01	short	"VOLTage:AC"	
OFFSET	0x02	long	"FREQuency:OFFSet"	
FSTRENGTH	0x04	short	"FSTRength"	mit EB200FS
AM	0x08	short	"AM"	nur ESMB
AM_POS	0x10	short	"AM:POSitive"	nur ESMB
AM_NEG	0x20	short	"AM:NEGative"	nur ESMB
FM	0x40	long	"FM"	nur ESMB
FM_POS	0x80	long	"FM:POSitive"	nur ESMB
FM_NEG	0x100	long	"FM:NEGative"	nur ESMB
PM	0x200	short	"PM"	nur ESMB
BANDWIDTH	0x400	long	"BANDwidth"	nur ESMB
CHANNEL	0x00010000	unsigned short	"CHANnel",	
FREQUENCY	0x00020000	unsigned long	"FREQuency:RX“	
SWAP	0x20000000		"SWAP"	
SIGNAL_GREATER_SQUELCH	0x40000000	-	„SQUelch“	
OPTIONAL_HEADER	0x80000000	-	"OPTional"	

Tabelle 2: Beschreibung der SelectorFlags

Diese SelectorFlags beschreiben, welche Daten in PeriodicTraceData vorzufinden sind, ob ein OptionalHeader übertragen wurde, und ob es sich bei den Tracedaten um „SIGNAL_GREATER_SQUELCH“ Daten handelt. Wenn das SelectorFlag SWAP gesetzt ist, werden die Nutzdaten in der Little-Endian-Order übertragen. Die Reihenfolge der Daten in den PeriodicTraceData ist entsprechend der Reihenfolge der o.a. Tabelle, also, je nach gesetzten SelectorFlags.

Diese Flags werden bestimmt durch erstens die entsprechenden Konfigurationsbefehle "TRAC:UDP:.....", zweitens durch die aktuell eingestellten Sensor-Funktionen und drittens, ob die entsprechende Trace-Art diese Einstellung überhaupt zuläßt. Wenn all diese Einstellungen ein bestimmtes Datum erlauben, wird dieses im Datagramm-Trace geschickt und das entsprechende SelectorFlag gesetzt.

FScanTrace

In dieser Trace-Art sind alle in den SelectorFlags spezifizierten Daten relevant.

Beschreibung des OptionalHeaders:

```
OptionalHeader {
    short cycleCount;
    short holdTime;
    short dwellTime;
    short directionUp;
    short stopSignal;
    unsigned long startFrequency;
    unsigned long stopFrequency;
    unsigned long stepFrequency;
}
```

Beispiel eines vollständigen Attributes:

```
FScanAttribute {
    short number_of_trace_values;          /* entspricht number_of_trace_items */
    char reserved;
    unsigned char optional_header_length;  /* 0 oder 22 */
    unsigned long selectorFlags;           /* siehe Tabelle 2: Beschreibung der SelectorFlags */
    OptionalHeader;                        /* wie oben beschrieben, wenn optional_header_length
                                              = 22 */

    short level-1;
    short level-2;
    short level-3;
    ....
    short level-number_of_trace_values;

    long offset-1;
    long offset-2;
    long offset-3;
    ....
```

```
    long offset-number_of_trace_values;

    short am-1;
    short am-2;
    short am-3;
    ....
    short am-number_of_trace_values;

    /* usw. usw. */
    /* Reihenfolge siehe Tabelle SelectorFlags */

    ....
    unsigned long frequency-1;
    unsigned long frequency-2;
    unsigned long frequency-3;
    ....
    unsigned long frequency- number_of_trace_values;
}
```

Es ist möglich, dass in Zukunft Elemente zum OptionalHeader dazugefügt werden, genauer hinten angefügt werden. Sofern eine Applikation mit optional_header_length über eben diesen hinwegspringt um zu den Trace Daten zu gelangen, ergibt sich für bestehende Programme kein Problem (aufwärtskompatibel).

MScanTrace

Dieser Trace hat die gleiche Struktur wie der FScanTrace mit Ausnahme des OptionalHeaders, der einige Elemente des FScanTrace-OptionalHeaders nicht beinhaltet.

Beschreibung des OptionalHeaders.

```
OptionalHeader {
    short cycleCount;
    short holdTime;
    short dwellTime;
    short directionUp;
    short stopSignal;
}
```

Entsprechend ist optional_header_length entweder 0 oder 10.

DScanTrace

SelectorFlags für DScanTrace kann enthalten:

LEVEL

FSTRENGTH

SIGNAL_GREATER_SQUELCH

OPTIONAL_HEADER

Beschreibung des OptionalHeaders.

```
OptionalHeader {
    unsigned long startFrequency;
    unsigned long stopFrequency;
    unsigned long stepFrequency;
    unsigned long markFrequency;
    short bwZoom;
    short referenceLevel;
    short newStepScheme;        /* ab UDP Version 0x0224
}
```

Ab *minor_version_number 0x24* wird beim DSCAN-Daten im *OptionalHeader* das zusätzliche Flag *newStepScheme* übertragen. Dieses Flag zeigt, ob das alte oder das neue Kanalraster (siehe auch Anhang J) gewählt wurde.

AUDio

SelectorFlags für Audio Daten kann nur OPTIONAL_HEADER enthalten.

Beschreibung des OptionalHeaders.

```
OptionalHeader {
    short audio_mode;                   /* siehe Remotebefehl SYSTem:AUDio:REMote:MODe
                                           */
    short frame_len;                    /* beschreibt die Anzahl der Bytes pro Frame */
    unsigned long frequency;            /* momentane Empfangsfrequenz */
    unsigned long bandwidth;            /* momentane ZF-Bandbreite */
    unsigned short demodulation;        /* momentane Demodulationsart */
    char    sDemodulation[8];           /* Demodulationsart als String */
}
```

Die verschiedenen Demodulationsarten sind beim EB200 folgendermaßen als Enum kodiert:

FM	0
AM	1
PULS	2
CW	3
USB	4
LSB	5
IQ	6

Ab *minor_version_number 0x26* wird bei AUDIO ein Demodulationsstring mitgeschickt.

Beispiel eines vollständigen Attributes:

```
AudioAttribute {
    short number_of_frames;             /* entspricht number_of_trace_items */
    char reserved;
    unsigned char optional_header_length;
    unsigned long selectorFlags;

    OptionalHeader;                     /* wie oben beschrieben */

    unsigned char data[frame_len * number_of_frames];
```

Die NF-Datenpakete werden intern mit einem Zyklus von 30 ms weitergereicht. Die Größe der UDP Pakete beträgt je nach audio_mode zwischen 325 Bytes und 4 kBytes.

Jedes UDP Paket enthält mehrere komplette Frames. Die Definition des audio_mode ist beim Remotebefehl SYSTem:AUDio:REMote:MODe beschrieben..

In den PCM –Modi (audio_mode 1 bis 12) enthält ein Frame ein oder zwei Kanäle und jeder Kanal ist 8 oder 16 Bit breit. Damit enthält ein Frame je nach Konfiguration 1, 2 oder 4 Bytes.

Im GSM 6.10 Mode (audio_mode 13) enthält ein Frame 65 Bytes. Dieser 520 Bit lange Frame enthält zwei 260 Bit lange GSM Teilframes. Die Bedeutung der Bits ist der ETSI Norm (www.etsi.org) zu entnehmen.

Byte:

0	1							32						63	64
Erster ETSI / GSM 6.10 Teilframe								Zweiter ETSI / GSM 6.10 Teilframe							

Anordnung der Bits und Bytes in einem Frame:

		Bit-Bezeichnung entsprechend ETSI	**Bits im Frame**	**Bytes im Frame**
1. Teilframe	LSB	1	0	0
		2	1	0
		3	2	0
		4	3	0
		5	4	0
		6	5	0
		7	6	0
		8	7	0
		9	0	1
		10	1	1
		11	2	1
		.	.	.
		.	.	.
		.	.	.
		254	5	31
		255	6	31
		256	7	31
		257	0	32
		258	1	32
		259	2	32
	MSB	260	3	32
2. Teilframe	LSB	1	4	32
		2	5	32
		3	6	32
		4	7	32
		5	0.	33.
		.	.	.
		.	.	.
		257	4	64
		258	5	64
		259	6	64
	MSB	260	7	64

Tabelle 3: Beschreibung des GSM Datenformats

Das SelectorFlag „SWAP“ hat keine Auswirkung auf die Anordnung der Bytes im GSM 6.10 Frame.

IFPan

SelectorFlags für ZF-Panorama muss enthalten:
LEVEL
OPTIONAL_HEADER

Beschreibung des OptionalHeaders.

```
OptionalHeader {
    unsigned long frequency;
    unsigned long spanFrequency;
    short averageTime;
    short averageType;
}
```

FASTLEVCW

SelectorFlags für diese Betriebsart muss LEVEL enthalten:
Es existiert kein OPTIONAL_HEADER.

LIST

SelectorFlags für diese Betriebsart muss LEVEL enthalten:
Es existiert kein OPTIONAL_HEADER.

CW

In dieser Trace-Art sind wie bei FScanTrace alle in den SelectorFlags spezifizierten Daten relevant.

Beschreibung des OptionalHeaders:

```
OptionalHeader {
    unsigned long Frequency;
}
```

Remote Befehle

Es ist eine feste Anzahl von maximal konfigurierbaren UdpPath Einträgen vorhanden. Der erste Eintrag repräsentiert dabei immer den Default Eintrag. Dieser Eintrag wird im CMOS RAM abgespeichert und bleibt über einen Aus-/Ein-Schaltzyklus erhalten. **Das bedeutet, daß auch nach dem Ausschalten (und folgendem Einschalten) initiierte Scan-Vorgänge Datagramm Pakete produzieren können, "ohne dass man es merkt".**

Ein UdpPath besteht immer aus IP-Addresse (als String) und Port Nummer (als Integer):

z.B.: "192.168.1.1", 18457

Byte-Order der zu übertragenden Daten:

Die Default Byte-Order ist Big-Endian (native Byte-Order des Gerätes), die auch der Netzwerk-Byte-Order entspricht, wie sie zum Beispiel im TCP/IP Protokoll Verwendung findet. Über das weiter unten beschriebene Flag „SWAP“ kann dieses Verhalten geändert werden. Sollte dieses Flag eingeschaltet sein, werden die Nutzdaten in der Little-Endian-Order übertragen. Dies betrifft sowohl die diversen OptionalHeaders als auch die PeriodicTraceData, nicht jedoch die „darüberliegenden“ Daten, da diese Protokollcharakter haben. Das Format der GSM 6.10 Audiodaten bleibt ebenfalls unberührt von dem SWAP Flag.

Registrieren von TAGs für einen bestimmten UdpPath:

TRACe:UDP:TAG[:ON] IP-ADDR, PORT-NUM, tag [, tag ..]

TRACe:UDP:DEFault:TAG[:ON] IP-ADDR, PORT-NUM, tag [, tag ..]

Die erste Form registriert einen beliebigen UdpPath, die zweite Form den Default UdpPath (Index 0).

Mögliche Tags:

FSCan, MSCan, DSCan, AUDio, IFPan, FASTlevcw, LIST, CW

Beispiel:

TRAC:UDP:DEF:TAG "89.10.20.30", 17222, FSC, MSC

Registrieren von FLAGs für einen bestimmten UdpPath:

TRACe:UDP:FLAG[:ON] IP-ADDR, PORT-NUM, flag [, flag ..]

TRACe:UDP:DEFault:FLAG[:ON] IP-ADDR, PORT-NUM, flag [, flag ..]

Die erste Form registriert einen beliebigen UdpPath, die zweite Form den Default UdpPath (Index 0).

Mögliche Flags	*Datenausgabe*	*Bemerkung*
"VOLTage:AC"	Pegel	
"FREQuency:OFFSet"	Ablage	
"FSTRength"	Feldstärke	mit EB200FS
"AM"	AM-Modulationsgrad	nur ESMB
"AM:POSitive"	AM-positiver Modulationsgrad	nur ESMB
"AM:NEGative"	AM-negativer Modulationsgrad	nur ESMB
"FM"	Frequenzhub	nur ESMB
"FM:POSitive"	Positiver Frequenzhub	nur ESMB
"FM:NEGative"	Negativer Frequenzhub	nur ESMB
"PM"	Phasenhub	nur ESMB
"BANDwidth"	Bandbreite	nur ESMB
"CHANnel",	Kanalnummer	
"FREQuency:RX“	Frequenz	
"SWAP"	in Little-Endian-Order	
„SQUelch“	nur Pegelwerte, die über der Squelchschwelle liegen	
"OPTional"	Zusätzlicher Optional Header	

Tabelle 4: Beschreibung der Flags

Hinweis:
Die Sensorfunktion “FSTRength“ liefert nur Ergebnisse, wenn die Softwareoption EB200FS bestückt ist. Siehe auch Anhang H.

Bemerkung: Die Flags werden unabhängig von den Tags registriert.

Beispiel:

TRAC:UDP:FLAG "89.255.255.255", 18457, "AM", "AM:POSitive", "AM:NEGative", "OPT"

De-Registrieren von TAGs für einen bestimmten UdpPath:

TRACe:UDP:TAG:OFF IP-ADDR, PORT-NUM, tag [, tag ..]
TRACe:UDP:DEFault:TAG:OFF IP-ADDR, PORT-NUM, tag [, tag ..]

Tags wie bei der Registrierung.

Beispiel:
TRAC:UDP:DEF:TAG:OFF "89.10.20.30", 17222, FSC

De-Registrieren von FLAGs für einen bestimmten UdpPath:

TRACe:UDP:FLAG:OFF IP-ADDR, PORT-NUM, flag [, flag ..]
TRACe:UDP:DEFault:FLAG:OFF IP-ADDR, PORT-NUM, flag [, flag ..]
Flags wie bei der Registrierung.

Beispiel:
TRAC:UDP:FLAG:OFF "89.255.255.255", 18457, "OPT"

Löschen eines UdpPath:

TRACe:UDP:DELete IP-ADDR, PORT-NUM

Löscht einen UdpPath von der Liste, sofern er gefunden werden kann. Auch der Default UdpPath kann auf diese Weise gelöscht werden.

Beispiel:

TRACE:UDP:DELETE "89.255.255.255", 18457

Löschen aller UdpPaths:

TRACe:UDP:DELete ALL

Löscht alle UdpPaths

Beispiel:

TRACE:UDP:DELETE ALL

Abfrage von UdpPath:

TRACe:UDP? MINimum | MAXimum | DEFault

Abfrage des Indexes des ersten UdpPath, des höchsten UdpPaths und des Default UdpPaths.

TRACe:UDP? <numeric_value>

Abfrage des UdpPaths mit dem Index <numeric_value>

Beispiel:

TRAC:UDP? MAX	->	3
TRAC:UDP? 0	->	DEF "89.10.20.30", 18457, FSC, MSC, "VOLT:AC", "OPT"
TRAC:UDP? 3	->	003 "255.255.255.255", 17222, DSC, "VOLT:AC", "OPT"