Device Management Schnittstellen

Referat von Peter Voser
Embedded Development GmbH

Winterthur, 31. Mai 2016

swissT.meeting
Embedded Computing Conference

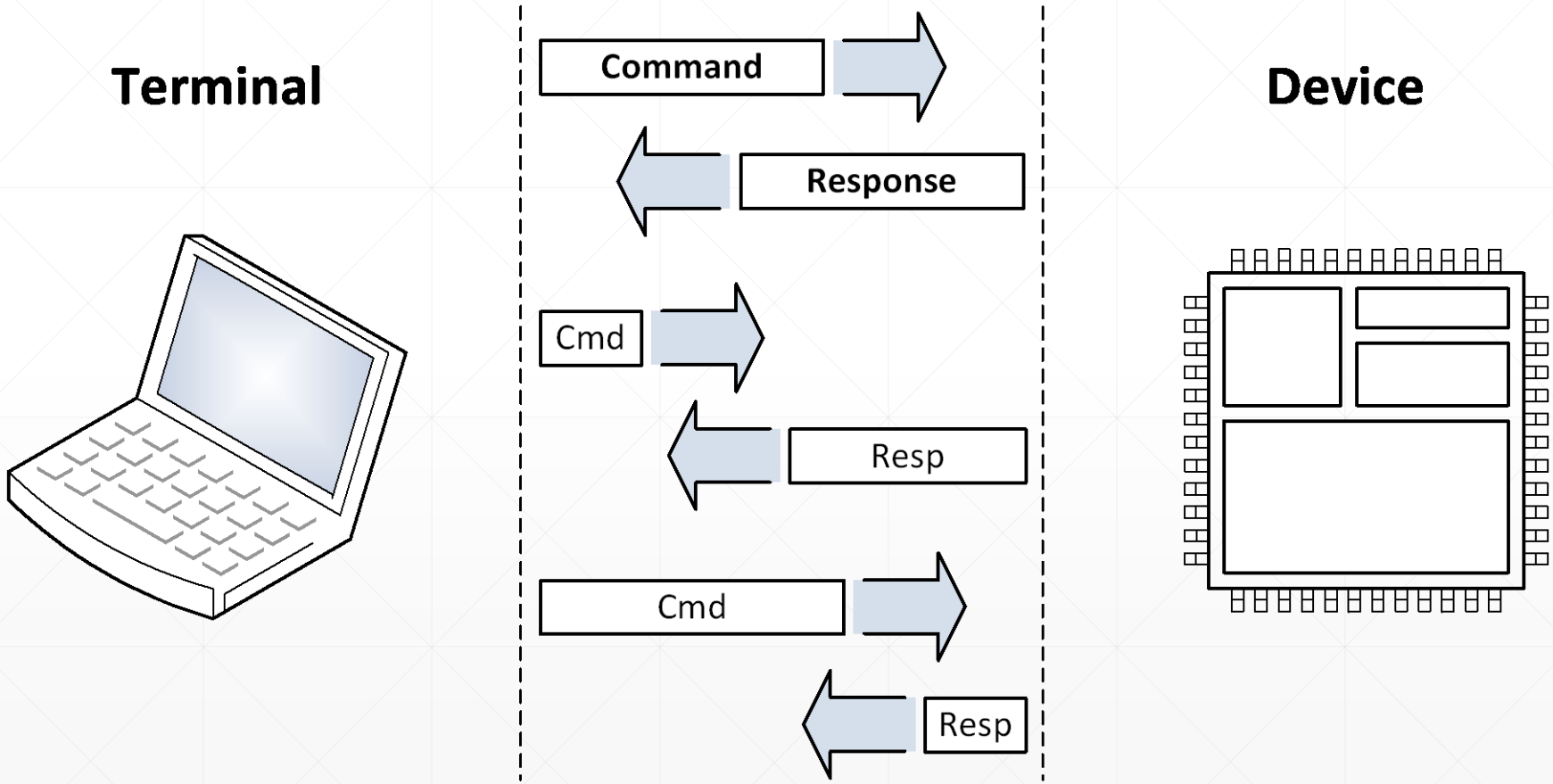
Device Management ist…

- Gerätesteuerung
- Parametrisierung
- Zugang zu internen Messgrössen und Zuständen
- Software Upgrade

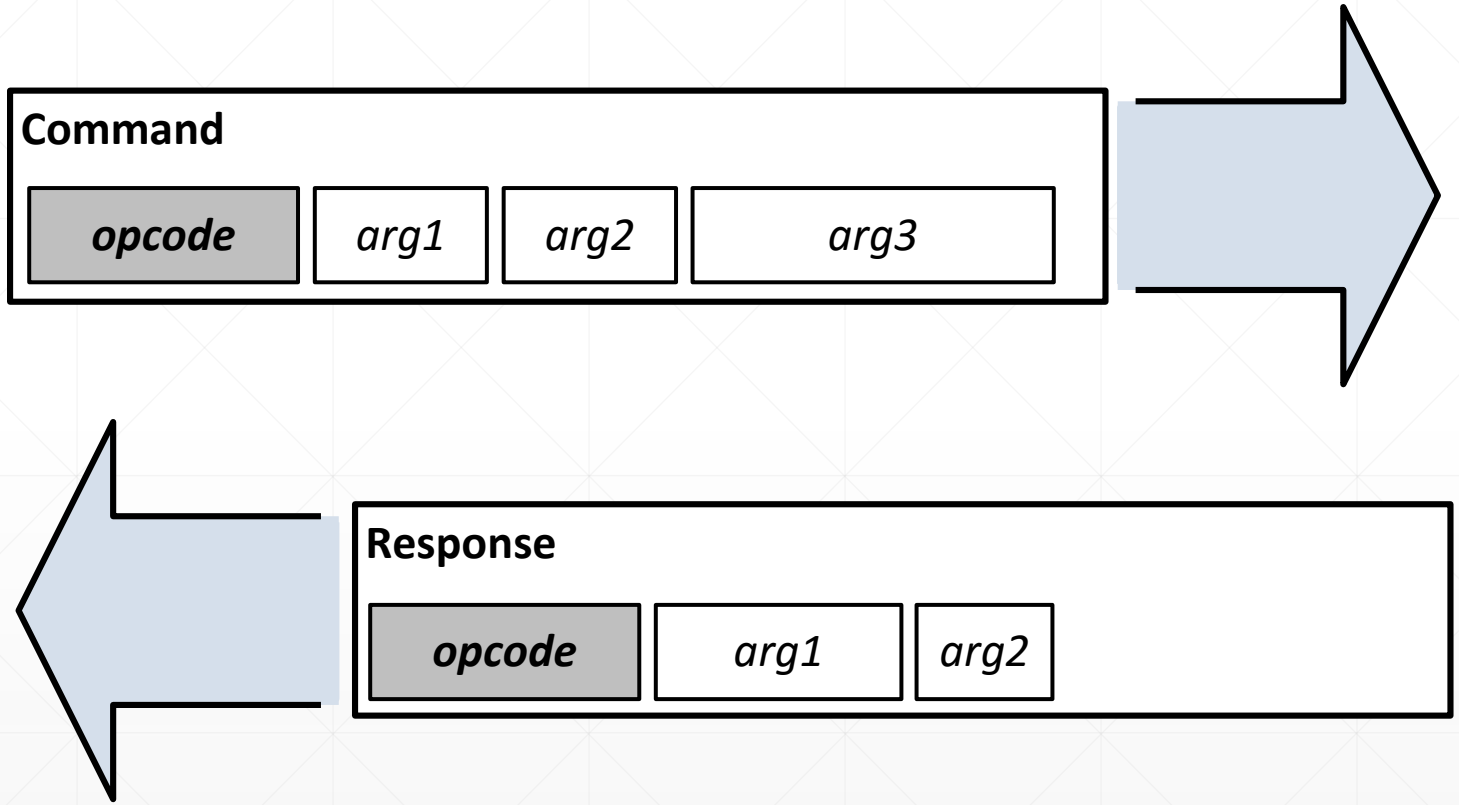
Embedded Development GmbH ist…
Engineering and Development

- Embedded Software Entwicklung
- Hardware/Software Co-Design
- Erfahrung aus diversen Branchen

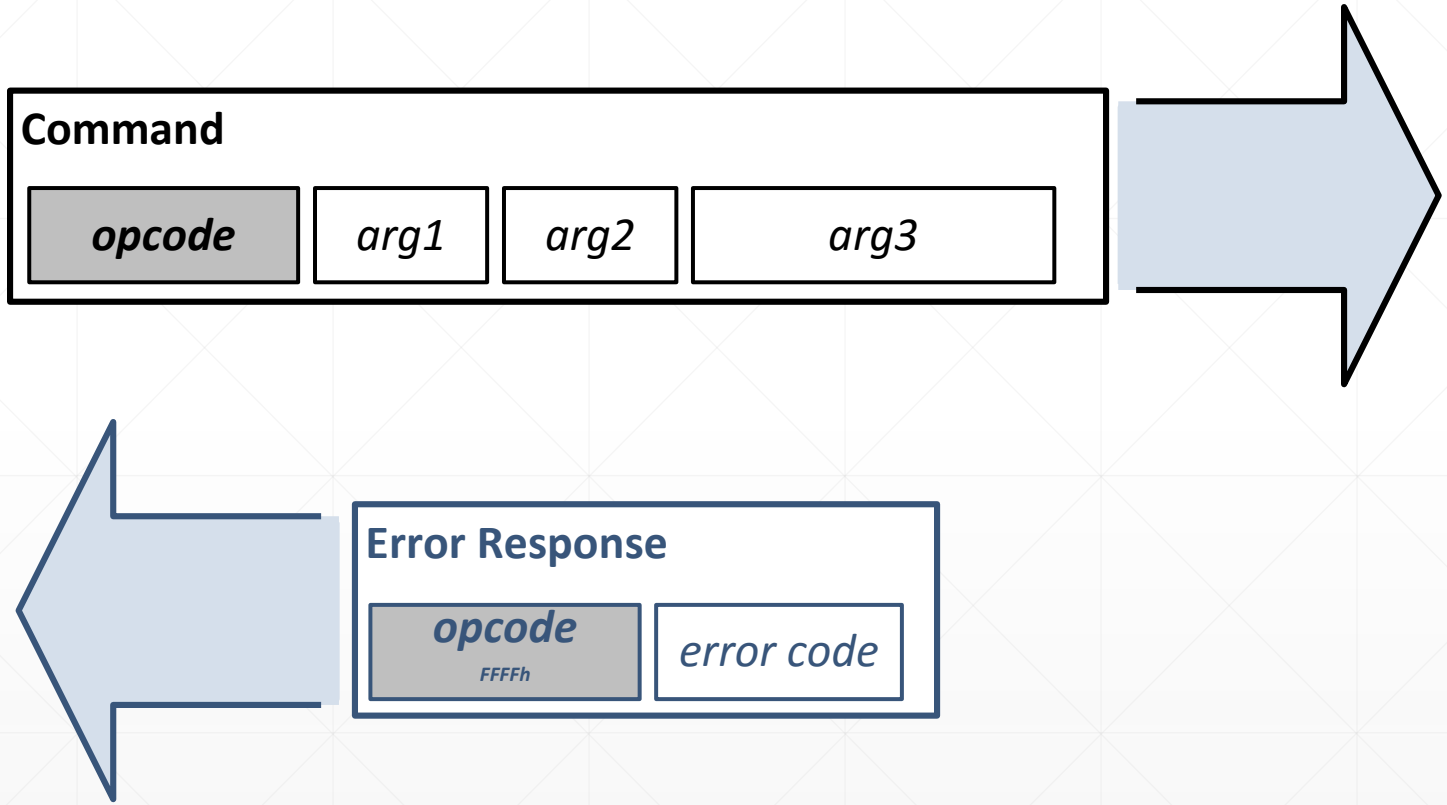
Messages
Das Basis Design



Messages
Das Basis Design



Messages
Fehler über generische Error Response



USB als Transport
USB Endpunkte

	Control	Bulk	Interrupt	Isochronous
Garantierte Bandbreite	Low/Full: 10% High/Super: 20%	✘	Low/Full: 90% High/Super: 80%	
Max. Paketgrösse	Low: 8 Full: 64 High: 64 Super: 512	Low: ✘ Full: 64 High: 512 Super: 1024	Low: 8 Full: 64 High: 1024 Super: 1024	Low: ✘ Full: 1023 High: 1024 Super: 1024
Garantierte Pakete/Zeit	✘	✘	Low: 1 / 10 ms Full: 1 / ms High: 3 / 125 µs Super: 3 / 125 µs	Low: ✘ Full: 1 / ms High: 3 / 125 µs Sup.: 48 / 125 µs
Fehlerkorrektur	✓	✓	✓	✘

TCP/IP als Transport
Der *lwIP* Stack

- *lwIP* (*lightweight IP*) ist ein Open Source TCP/IP Stack für Embedded Systeme
- Unterstützte Protokolle: ARP, IP, ICMP, UDP, TCP, DNS, DHCP, …
- Schonender Umgang mit Ressourcen, konfigurierbar per config-Include File
- Ideal für SoCs mit integriertem Ethernet MAC

TCP Transmission Control Protocol
Sorgt für Datenintegrität

- **Verbindungsorientiert**

  - *Client* („Terminal“) muss zu *Server* („Device“) verbinden vor Datenaustausch

- **Unterteilung der Daten in Segmente**

  - Müssen vom Empfänger bestätigt werden (*acknowledgment*)
  - *Retranmission* beim Ausbleiben der Bestätigung
  → *reliable byte stream*

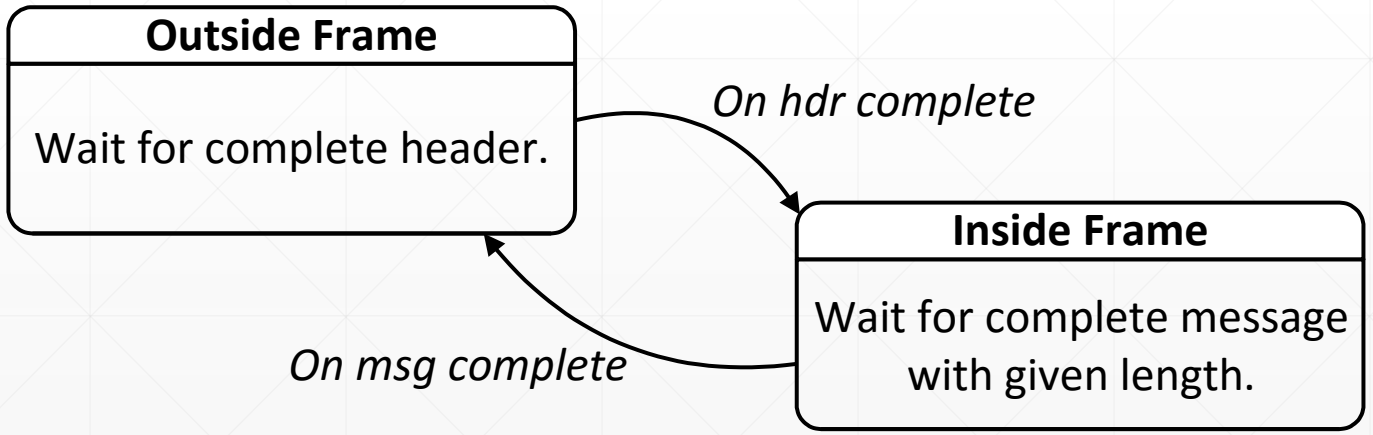
- **Datenintegrität**

  - Prüfsumme über TCP Header und Daten

→ lwIP: TCP ergibt ~50% der *object code size*

Messages
Framing

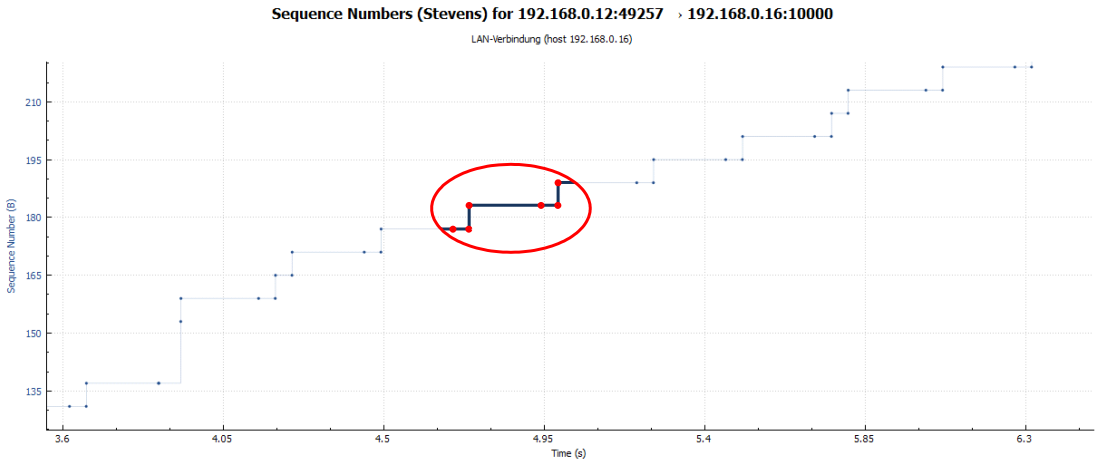
- **Der Rahmen um eine Message ist durch den Transport nicht gegeben.**



TCP Nagle Algorithmus

- Kleine Pakete können Netzwerk verstopfen (eher bei WANs als bei LANs)
  Bsp.: 4 Bytes senden = 20 Bytes TCP Header + 20 Bytes IP Header = 44 Bytes
- Algorithmus: wenn eine TCP Verbindung unbestätigte Daten hat, können keine kleinen Segmente geschickt werden, bis die Daten bestätigt sind.
- Kleines Segment: Segment < MSS
  Max. segment size = 1500 – 40 = 1460 Bytes bei Ethernet

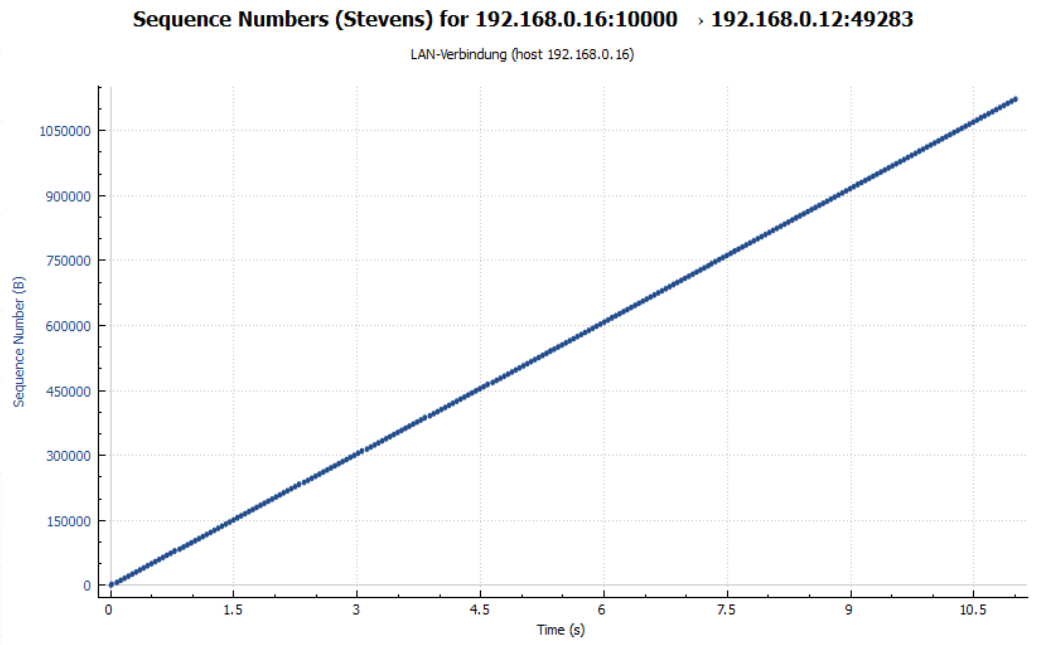
TCP Nagle Algorithmus mit lwIP



> 200 msec

No.	Time	Source	Destination	Protocol
127	*REF*	192.168.0.12	192.168.0.16	TCP
128	0.000359	192.168.0.16	192.168.0.12	TCP
129	0.000492	192.168.0.16	192.168.0.12	TCP
130	0.000519	192.168.0.12	192.168.0.16	TCP
131	0.000654	192.168.0.16	192.168.0.12	TCP
132	0.218334	192.168.0.12	192.168.0.16	TCP
133	0.218509	192.168.0.16	192.168.0.12	TCP
134	0.265131	192.168.0.12	192.168.0.16	TCP
135	0.265511	192.168.0.16	192.168.0.12	TCP
136	0.265683	192.168.0.16	192.168.0.12	TCP

`tcp_nagle_enable(pcb)` (default)



`tcp_nagle_disable(pcb)`

TCP Keepalive

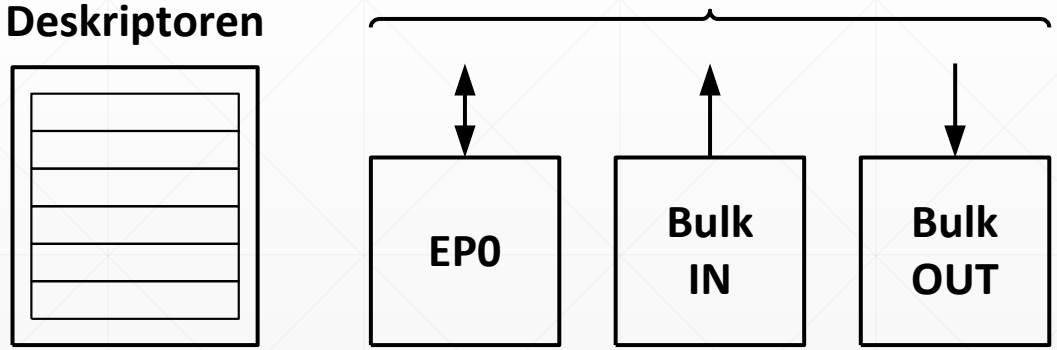
- Über eine inaktive TCP Verbindung findet kein Datenaustausch statt
- Etablierte Verbindung zu einem Server bleibt erhalten bis Close
- Deshalb stellen TCP Implementierungen *Keepalive* zur Verfügung
- Mittels Timer überprüft der Stack Inaktivität der Verbindungen
  → Senden von *Probe* Segmenten bei Inaktivität
- Nötig für Management Schnittstelle damit das Device (= TCP Server) die Ressourcen wieder freigeben kann

```
186 #if LWIP_TCP_KEEPALIVE
187   newPcb->so_options |= SOF_KEEPALIVE;
188   newPcb->keep_idle = 5000;
189   newPcb->keep_intvl = 5000;
190   newPcb->keep_cnt = 3;
191 #endif // LWIP_TCP_KEEPALIVE
```

USB als Transport
Vendor Class

- Selbstdefinierte Geräte
- Device Management Schnittstelle:

  - 1 Bulk OUT Endpunkt für Message Senden
  - 1 Bulk IN Endpunkt für Message Empfangen



USB als Transport
Verifikation mit USB Analysator (on wire)

Sp	m:s.ms.us	Len	Err	Dev	Ep	Record	Summary
	0:00.000.000					Capture started (Aggregate)	[02/10/16 19:19:31]
	0:00.000.545					<Manual Trigger>	
FS	0:04.676.426	18 B		12	00	Get Device Descriptor	Index=0 Length=18
FS	0:04.676.759	32 B		12	00	Get Configuration Descriptor	Index=0 Length=255
FS	0:04.676.759	8 B		12	00	SETUP txn	80 06 00 02 00 00 FF 00
FS	0:04.676.836	16 B		12	00	IN txn	09 02 20 00 01 01 00 80 32 09 04 00 00 02 FF 00
FS	0:04.676.866	16 B		12	00	IN txn	00 00 07 05 81 02 40 00 00 07 05 02 02 40 00 00
FS	0:04.676.891	0 B		12	00	IN txn	
FS	0:04.676.922	0 B		12	00	OUT txn	
FS	0:04.677.941	4 B		12	00	Get String Descriptor	Index=0 Length=255
FS	0:04.678.621	18 B		12	00	Get String Descriptor	Index=2 Length=255
FS	0:04.686.836	18 B		12	00	Get Device Descriptor	Index=0 Length=18
FS	0:04.687.605	9 B		12	00	Get Configuration Descriptor	Index=0 Length=9
FS	0:04.688.594	32 B		12	00	Get Configuration Descriptor	Index=0 Length=32
FS	0:04.689.585	2 B		12	00	Get Device Status	
FS	0:04.695.099	0 B		12	00	Set Configuration	Configuration=1
FS	0:09.470.336	6 B		12	02	OUT txn	02 00 FD FF 00 F0
FS	0:09.471.762	64 B		12	01	IN txn	50 00 AF FF 00 F0 03 00 0A 00 15 11 00 42 75 69 6C 64 20 69 6E 66 6F 72…
FS	0:09.471.836	20 B		12	01	IN txn	54 65 6D 70 65 72 61 74 75 72 65 20 32 03 00 C2 B0 43 02 01
FS	0:09.479.764	6 B		12	02	OUT txn	02 00 FD FF 03 F0
FS	0:09.479.764	3 B		12	02	OUT packet	E1 0C 71
FS	0:09.479.768	9 B		12	02	DATA1 packet	4B 02 00 FD FF 03 F0 FF 2E
FS	0:09.479.775	1 B		12	02	ACK packet	D2
FS	0:09.479.988	10 B		12	01	IN txn	06 00 F9 FF FF FF 06 00 00 00

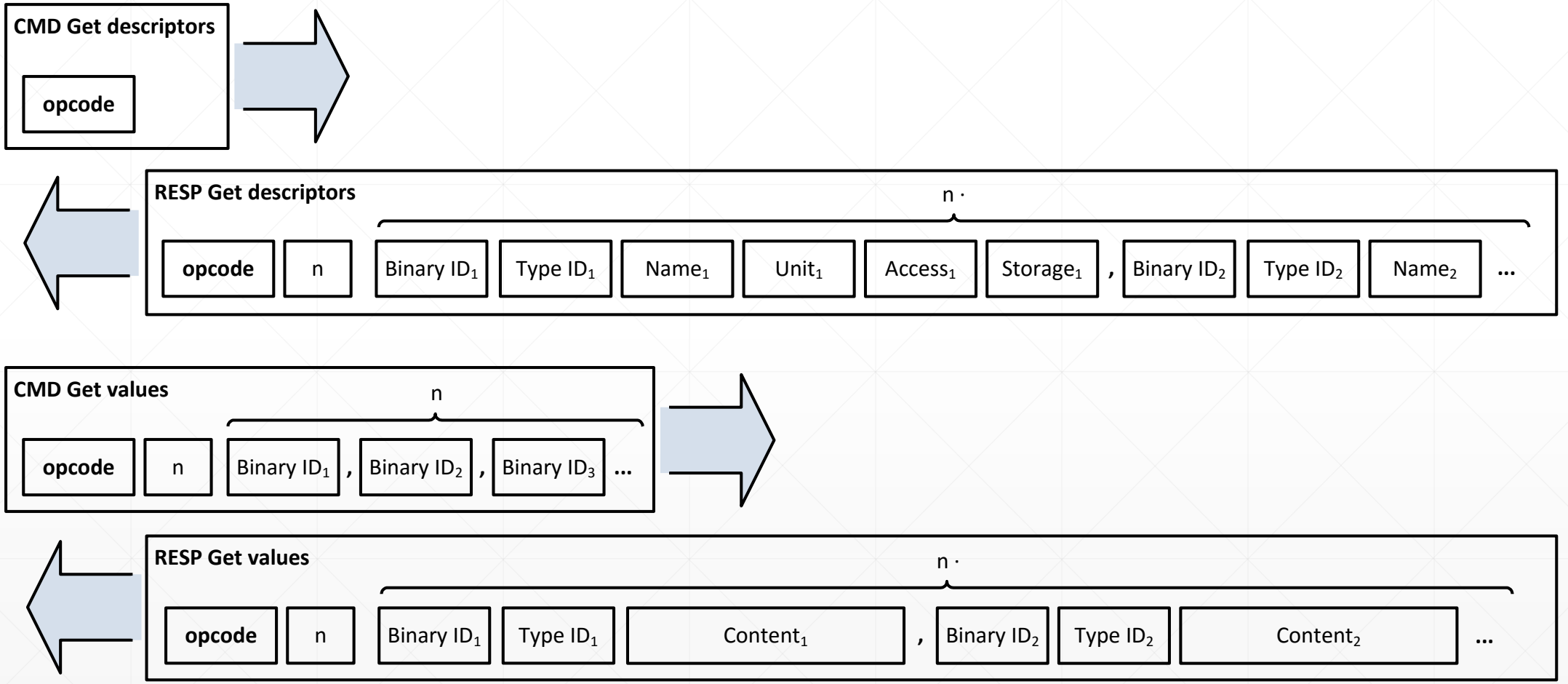
EP #1 = Bulk IN
EP #2 = Bulk OUT

Generische Parameter Beschreibungen, Datentypen

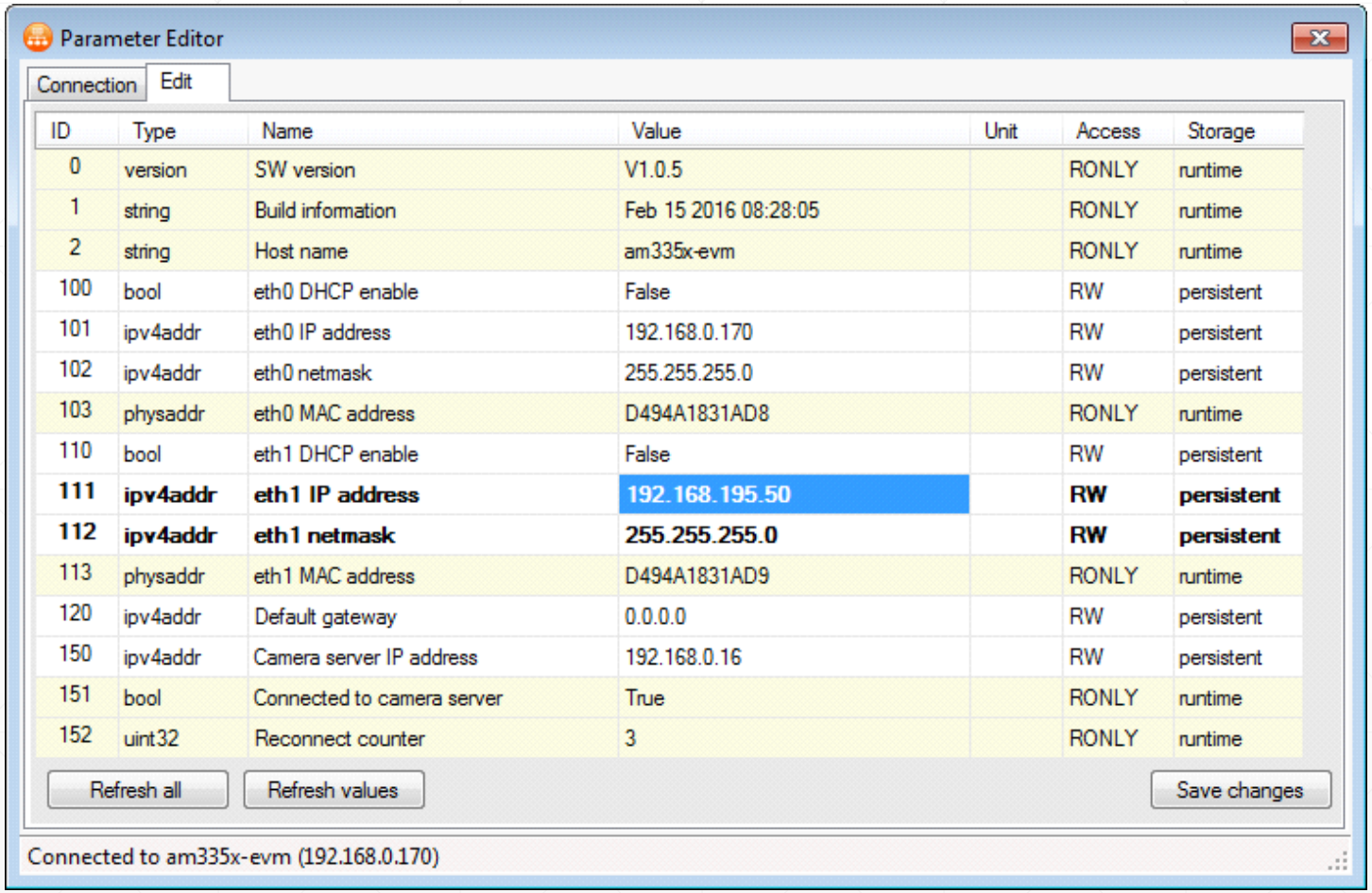
- Parameter haben abfragbare Beschreibungen (Deskriptoren)
- Parameter haben Datentypen, zum Beispiel:

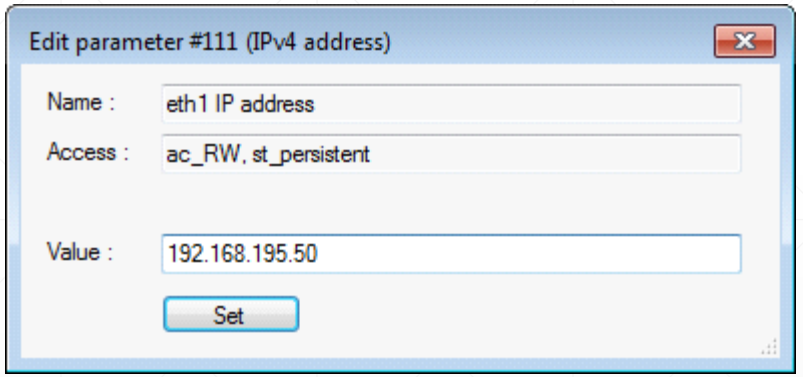
C99 Integers `#include <stdint.h>` uint8_t, int8_t, uint16_t, int16_t, uint32_t, int32_t, uint64_t, int64_t	**IEEE754 Floating Point** Single precision (32-bit) Double precision (64-bit)	**Kommunikation** IPv4 Adresse Physikalische Adresse
Nützliches Boolean String Version GUID (128-bit oder 96-bit)	**Zeit** DateTime TimeSpan	**...**

Generische Parameter
Die Schnittstelle



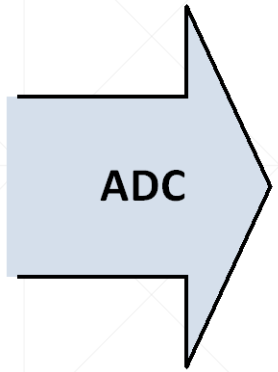
Generische Parameter
Der Editor

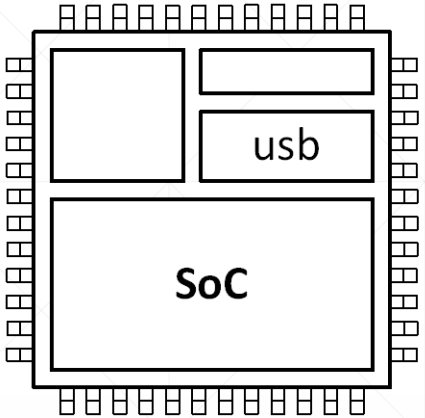


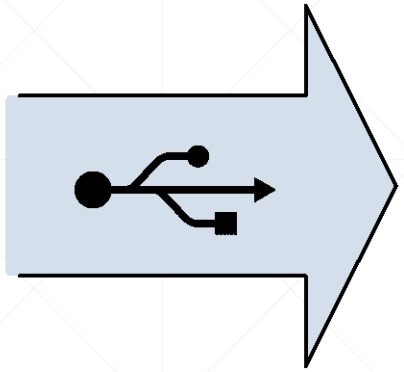


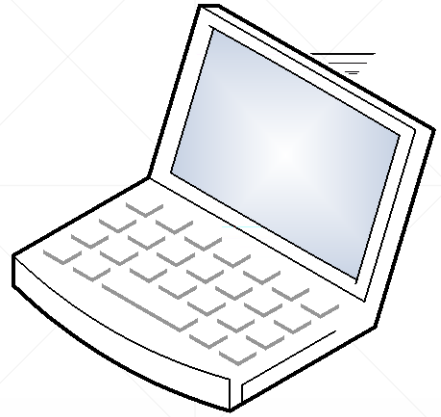
Demo 1
USB T-Sensor

2x NTC T-Sensor

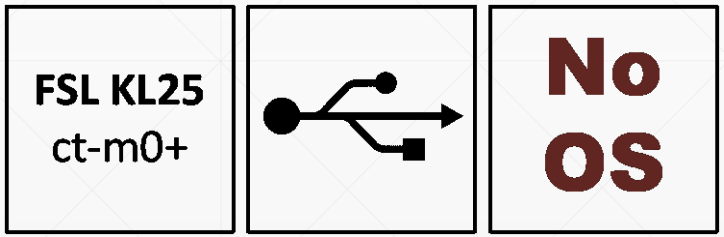




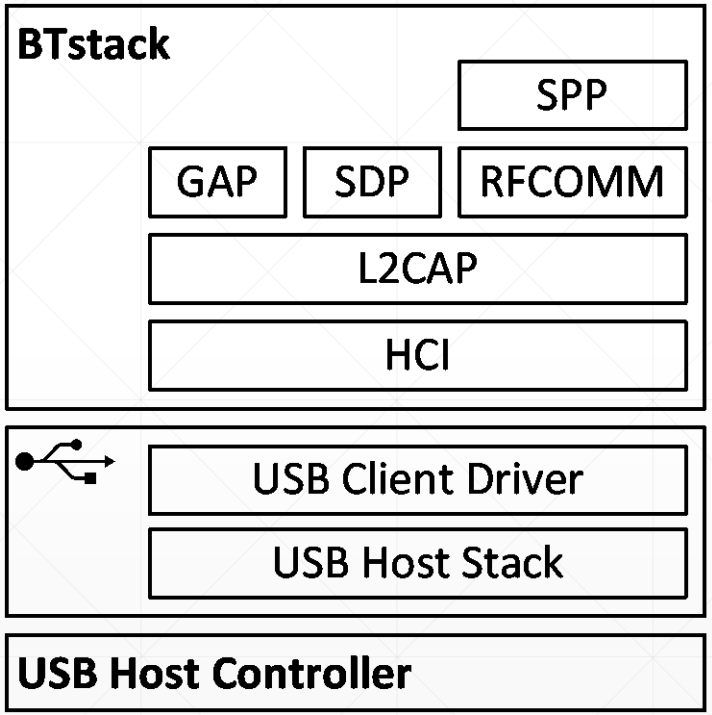




ID	Type	Name	Value	Unit	Access	Storage
0	version	SW version	V1.0.0		RONLY	runtime
1	string	Build information	Feb 19 2016 21:45:57		RONLY	runtime
20	bool	Sensor 1 present	True		RONLY	runtime
21	int16	Temperature 1	189	0.1 °C	RONLY	runtime
22	bool	Sensor 2 present	True		RONLY	runtime
23	int16	Temperature 2	189	0.1 °C	RONLY	runtime

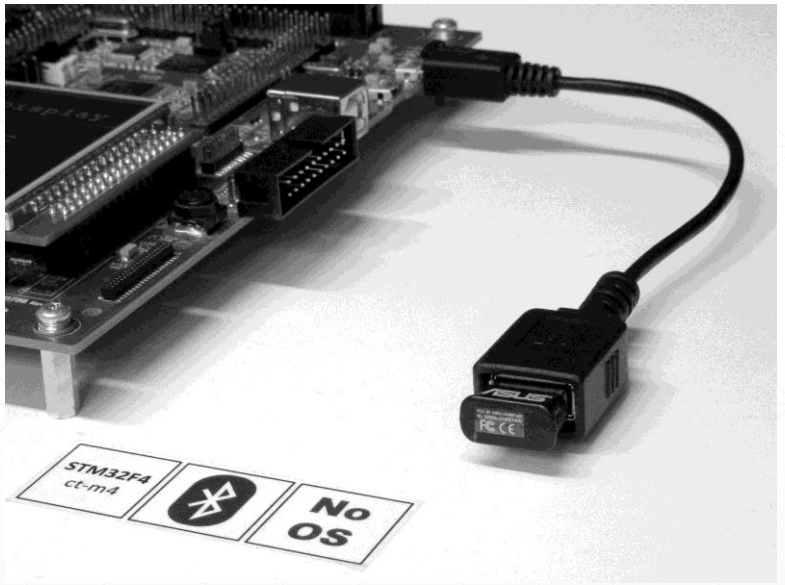


Demo 2
Bluetooth Display



blue kitchen

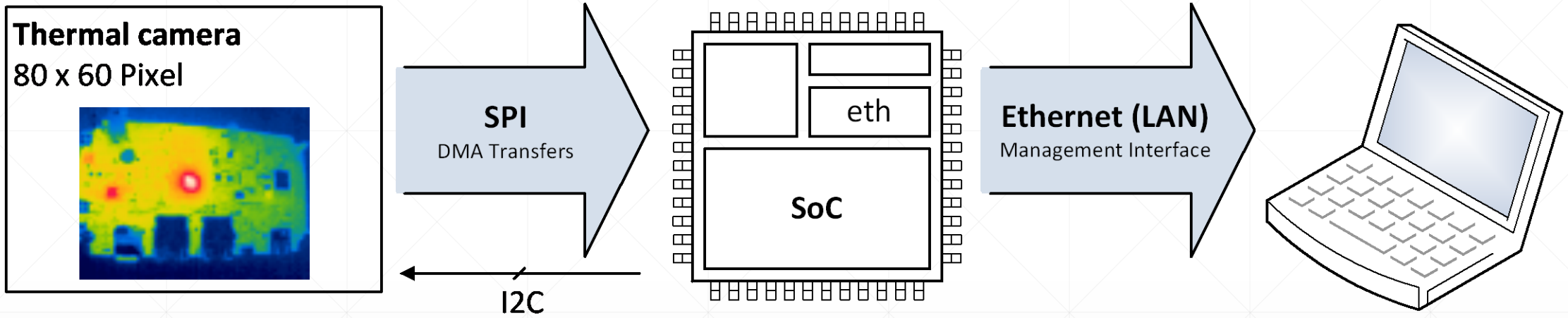
- Portierbarer BT Stack
- Für HCI Chipsets
- Bluetooth SIG qualifiziert



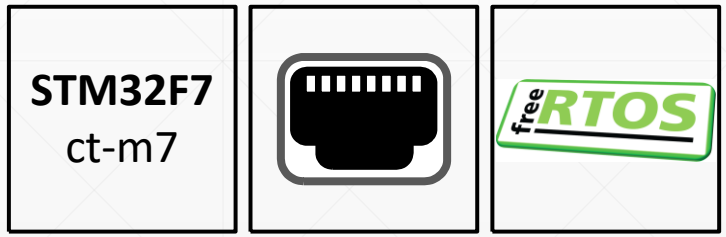
STM32F4
ct-m4

No
OS

Demo 3
Wärmebild Kamera über LAN

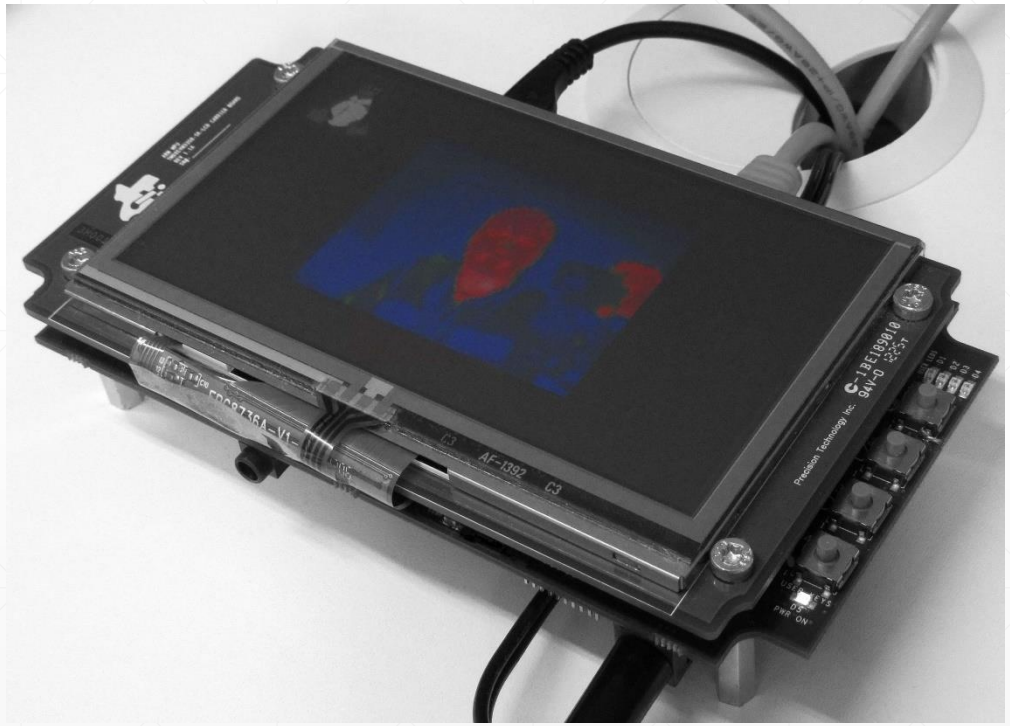


ID	Type	Name	Value	Unit	Access	Storage
0	version	SW version	V1.0.0		RONLY	runtime
1	string	Build information	Feb 29 2016 17:26:24		RONLY	runtime
20	float32	Camera frame rate	25.94034	fps	RONLY	runtime
21	uint32	Camera uptime	1599893	msec	RONLY	runtime
22	float32	T_aux	23.19	°C	RONLY	runtime
23	float32	T_fpa	27.41	°C	RONLY	runtime

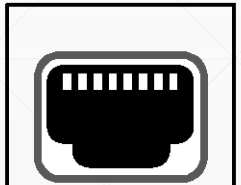


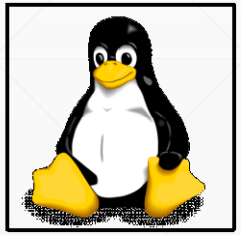
Demo 4
Terminal für Wärmebild Kamera über LAN

- Terminal: Client für Wärmebild Kamera Server



TI
AM335x
ct-a8





Fragen?

Embedded Development GmbH

www.embedded-development.ch

Peter Voser

077 405 70 05 Mobile

peter.voser@voser-development.ch