xCDT Sensor

SPI Specification

Writer	Checker	Approval	Name	Date
SAu	SAa	BeL	xCDT Sensor – SPI Specification	2024/05/27

Table of contents

1 xCDT Sensor - SPI Specification

--

1.1 History

Date	Modification	Author	Version
2023/02/17	Applicable to xCDT PR19 samples (Application 1.1.2.1) The SPI communication description is limited to the Application frames.	SAu	V6
2023/05/02	Applicable to xCDT PR24 samples (Application 1.1.3.5) - Add FallbackMode description about over/under temperature, over/under voltage and overcurrent faults - Update IntegrityFailMode description	SAu	V7
2024/05/27	Applicable to xCDT PR32 samples (Application 2.6.0.1) - Add Usage scenarios	SAu	V8

1.2 Glossary

CDT	Current Detection Transducer
CRC	Cyclic Redundancy Check
CT	Current Transformer
DCDT	Dual Current Detection Transducer (CDT + HF CT)
E2E	End To End
EVSE	Electric Vehicle Supply Equipment
HF	High Frequency
HW	Hardware
Iprim	Sum of primary currents on jumpers. Equal to zero Amp if no leakage current is present
LF	Low Frequency
LSB	Least Significant Bit
MCU	Micro Controller Unit
MISO	Master Input, Slave Output. Generated by SPI Slave. Corresponds to SDO (Serial device out) from MCU point of view

MOSI	Master Output, Slave Input. Generated by SPI Master. Corresponds to SDI (Serial device in) from MCU point of view
NSF	Not Safety Relevant
NVM	Non volatile memory (e.g. Flash)
OBC	On-Board Charger ; It contains the Host SPI Master, which communicates with the xCDT
POC	Proof Of Concept
POR	Power On Reset
RCD	Residual Current Detection
RCM	Residual current Monitoring
Rx	Reception, from Sensor to Master
SCLK	SPI Serial Clock. Generated by SPI Master
SF	Safety Function
SPI	Serial Peripheral Interface
Sps	Samples per second
/SS	Slave Select signal, active at low state. It is generated by SPI Master. (other terminology: CS Chip Select)
SW	Software
Tx	Transmission, from Master to Sensor
Vaux	Auxiliary voltage from Power supply
xCDT	CDT or DCDT sensor

1.3 Reference documents

Ref.	Document name	Version
[R1]	Bootloader SPI Interface protocol	V1
[R2]	xCDT Family Sensor – Application Note	V6

1.4 Document purpose

This document contains the SPI communication specification applicable to xCDT sensors loaded with software versions described in History chapter.

1.5 Usage scenarios

--

1.5.1 Pre-requisite

Product is downloaded with the bootloader (FlasherSw) and the application (MainSw).

Product is powered-on, with a Host connected on (SPI Master).

Product is running in RcdActiveMode.

1.5.2 Application scenarios

--

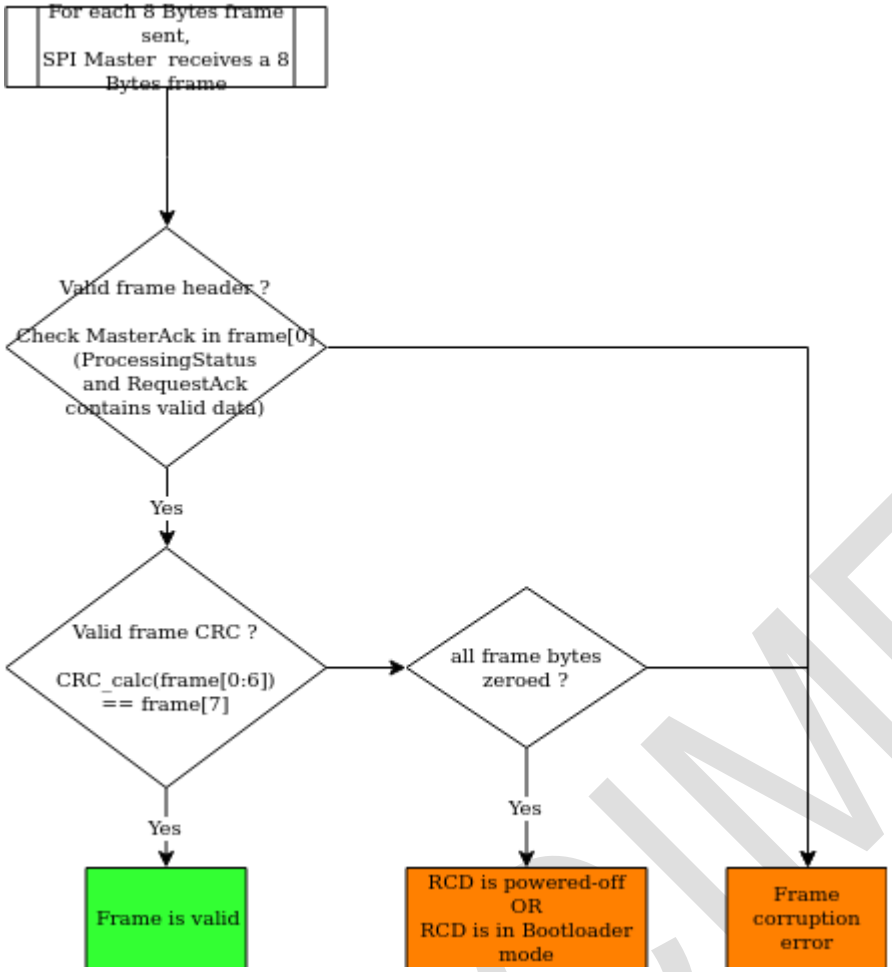
1.5.2.1 Establish a safety communication

To establish a safety communication:

- Send ApplicationFrame with a SPI frame rate of 1 KSps +/- 10%
- For each frame received from the product
  - Check Frame validity Rx frame verification to be done by Host
  - Check E2E counter validity E2E checking procedure by Host
- If no valid frames are received during more than the FHTI duration (duration is indicated in the Functional Safety Manual), the Host shall consider the safety communication is broken and take the appropriate action

1.5.2.1.1 Rx frame verification to be done by Host

Master shall check the validity of the SPI frame received before to process it.



1.5.2.1.2 E2E checking procedure by Host

The End-to-end counter values are described in E2eCounter. This mechanism allows the Master to survey the sensor capacity to compute fresh current samples.

The E2eCounter is initialized to 0 after xCDT reset, for its first response. The E2eCounter stays blocked at 0 until a HardwareInitModeRequest is received by the xCDT with an E2eInit between 1 and 254.

- ServiceModeRequest
- HardwareInitModeRequest (E2eInit)

After HardwareInitModeRequest(E2Einit) is received, the xCDT uses this E2EInit value and increments it from this value to 254, each time a new sample is calculated internally at fluxgate period (around 44 us). When the E2eCounter reaches the value 254, then E2eCounter restarts at 1 internally.

The value 255 indicates an E2E overflow situation, in case E2eCounter has reached the value 254 and the Master has not sent any ApplicationRequest frame since the E2E counter started from value 1. Then the E2eCounter remains in value 255 until the Master sends a HardwareInitModeRequest.

Another simpler way to start the E2eCounter incrementing is to set E2eInit using the byte 2 of

ApplicationRequest frame. Doing so doesn't oblige to go in ServiceMode in order to send the HardwareInitModeRequest.

To establish a safety communication from xCDT to Master, the sample rate for the Master requests shall be 1 KSps, i.e. a sending period of 1 ms. Then the Master shall check the E2E counter increment after each request:

- The Master sends an ApplicationFrameRequest at T0 time, and reads the E2E counter: E0
- The Master sends an ApplicationFrameRequest at T1 time, and reads E2E counter: E1
- The Master computes the maximum expected E2E increment: E2E_max_increment = int((T1 - T0) / 44 us)
    - Note: 44 microseconds corresponds to the average sample production rate (mean fluxgate period)
- The Master computes the tolerance for E2E increment: E2E_tolerance = max(1, (E2E_max_increment * 25 / 100))
    - Note: the variation is 100 * (55 - 44) / 44 = 25 %
- The Master asserts an error if the following condition is NOT respected: E2E_max_increment - E2E_tolerance <= modulo(E1 - E0, 254) <= E2E_max_increment + E2E_tolerance

Example for 1 KSps SPI rate: the E2E increment is 1 ms divided by around 0.044 ms (fluxgate bridge period) = 20..25 between each Host request.

1.5.2.2 Detect a tripping situation

While a safety communication is established,

- Read TripAC and TripDC fields from the ApplicationResponse frame
- If (TripAC is not Inactive) OR (TripDC is not Inactive)
    - Host SPI Master shall activate the safe state (e.g. open charging relays)

1.5.2.3 Read current leakage and temperature thresholds

Perform an ApplicationFrameRequest exchange to get CurrentCH1, CurrentCH2 and temperature range in ModuleData.

1.5.2.4 Diagnose fallback events

If ApplicationFrameRequest returns FallbackMode (in ModuleState), an external constraint (such as over/under temperature, over/under voltage or an overcurrent) has arised, as stated in ModuleData.

Remove the external constraint to have the Product returning to RcdActiveMode.

1.5.2.5 Diagnose integrity failure

If ApplicationFrameRequest returns IntegrityFailMode (in ModuleState), an internal error has been detected by the safety checks (at startup or at runtime) as stated in ModuleData.

Read the full error context (to help diagnostic by sending the content to LEM support):

- ReadFaultContextRequest

Note: an automatic reset is occuring after a timeout (0.5s); the command shall be send before it occurs.

1.5.2.6 Read temperature and power supply voltage

- ServiceModeRequest
- PrimaryMeasurement

1.5.2.7 Reduce power consumption

Enter low power mode

- LowPowerModeRequest

The method to exit the low power mode is described in LowPowerModeRequest.

1.5.2.8 Read product identification

- ServiceModeRequest
- Hw Product identification
- Sw Product identification

1.5.2.9 Update the product

- ServiceModeRequest
- FlasherModeRequest
- Send new application binary data: refer to **Bootloader SPI Interface protocol [R1]** documentation

1.6 Application Layer

--

1.6.1 Actions

--

1.6.1.1 ApplicationFrameRequest

- Send ApplicationRequest frame
- Receive ApplicationResponse frame

1.6.1.1.1 Nominal example

The SPI exchange for an ApplicationFrameRequest sent while the xCDT is in RcdActiveMode can be as following:

Note: each line corresponds to a SPI exchange; the numbers are in hexadecimal format.

Master	Master Decoding	xCDT	xCDT Decoding
A0 00 00 00 00 00 00 AD	HostCommand=b101 (ApplicationRequest), HostRequestCode=b00000 (NoHostRequest) HostServiceData=all zeroes CRC	80 40 00 20 06 20 00 25	ProcessingStatus=b100 (PositiveResponse), RequestAck=b00000 (NoHostRequest) ModuleState=b010 (RcdActiveMode), ModuleData=b00000: Temp<=56,5°C, coming from startup E2eCounter=0x00 TripDC=0, CurrentCH1=0x2006=0.6mA TripAC=0, CurrentCH2=0x2000=0.0mA CRC

1.6.1.2 ServiceModeRequest

- Send OperationRequest frame with HostRequest set to "Mode request, ServiceMode"
- Receive ServiceResponse frame

1.6.1.2.1 Nominal example

The SPI exchange for ServiceModeRequest sent while the xCDT is in RcdActiveMode can be as following:

Note: each line corresponds to a SPI exchange; the numbers are in hexadecimal format.

Master	Master Decoding	xCDT	xCDT Decoding
63 04 00 00 00 00 08 04	HostRequestCode=0x3=ModeRequest HostServiceData=0x04=ServiceMode Dummy value, not necessary CRC	80 40 60 20 0E 1F FD AD	ProcessingStatus=b100 (PositiveResponse), RequestAck=b00000 (NoHostRequest) ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) E2eCounter TripDC=0, CurrentCH1=0x200E=1.4mA TripAC=0, CurrentCH2=0x1FFD=-0.3mA

			CRC
A0 00 00 00 00 00 09 67		43 40 64 1F DC 1F FD 96	ProcessingStatus=b010=ResponsePending, RequestAck=0x3=ModeRequest (*) ModuleState=b010=RcdActiveMode, ModuleData=b00000 (No fault) RcdE2eCounter TripDC=0, CurrentCH1=0x1FDC=-3.6mA TripAC=0, CurrentCH2=0x1FFD=-0.3mA CRC
A0 00 00 00 00 00 0A 49		83 60 81 00 00 00 00 4D	ProcessingStatus=b100=PositiveResponse, RequestAck=0x3=ModeRequest ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) FirstFrameIndicator=1, DataSequenceIndex=0x01=frame number1 ServiceResponseData: None CRC

(*): At this stage, the RCD ModuleState is still RcdActiveMode, so the xCDT answer is an ApplicationResponse frame. See ServiceResponse frame processing

1.6.1.2.2 Example with change mode refused

The SPI exchange for a ServiceModeRequest sent while the xCDT is already in ServiceMode can be as following:

Note: each line corresponds to a SPI exchange; the numbers are in hexadecimal format.

Master	Master Decoding	xCDT	xCDT Decoding
63 04 00 00 00 00 00 59	ServiceModeRequest=0x63	80 60 C4 60 06 60 00 F2	ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) CRC
A0 00 00 00 00		C3 60 DC 60 06	ProcessingStatus=b110=ConditionsNotCorrect, RequestAck=0x3=ModeRequest ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) RcdE2eCounter

00 00 AD		5F FF BB	 TripDC, CurrentCH1 TripAC, CurrentCH2 CRC

1.6.1.3 HardwareInitModeRequest

- Send OperationRequest frame
  - HostRequest set to "Mode request, HardwareInitMode"
  - E2eInit set to a value between 1 and 254. An E2eInit equal to 0 or 255 has the same behavior than the value 1.
- Receive ServiceResponse frame

1.6.1.3.1 Nominal example

The SPI exchange for HardwareInitModeRequest sent while the xCDT is in ServiceMode can be as following:

Note: each line corresponds to a SPI exchange; the numbers are in hexadecimal format.

Master	Master Decoding	xCDT	xCDT Decoding
63 00 01 00 00 00 03 0A	HostRequestCode=0x3=ModeRequest HostServiceData=0x00=HardwareInitMode E2eInit=0x01 Dummy value, not necessary CRC	80 60 E0 1F F9 20 01 17	 ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) CRC
A0 00 00 00 00 00 04 48		43 60 E6 1F F9 1F FF A7	ProcessingStatus=b010=ResponsePending, RequestAck=0x3=ModeRequest ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) RcdE2eCounter TripDC, CurrentCH1 TripAC, CurrentCH2
A0 00 00 00 00 00 05 DF		83 40 81 00 00 00 00 C7	ProcessingStatus=b100=PositiveResponse, RequestAck=0x3=ModeRequest ModuleState=b010=RcdActiveMode (*), ModuleData=b00000 (No fault) FirstFrameIndicator=1, DataSequenceIndex=0x01=frame number1

			ServiceResponseData: None

(*) HardwareInitMode is a transient state, leading to RcdActiveMode.

1.6.1.4 FlasherModeRequest

- Send OperationRequest frame with HostRequest set to "Mode request, FlasherMode" and the parameter:
  - "Security Key, Flasher" = 0x94A3E8FF
- Receive ServiceResponse frame

The reset to activate the Bootloader in FlasherMode is done just after the PositiveResponse frame has been sent

1.6.1.4.1 Nominal example

The SPI exchange for FlasherModeRequest sent while the xCDT is in ServiceMode can be as following:

Note: each line corresponds to a SPI exchange; the numbers are in hexadecimal format.

Master	Master Decoding	xCDT	xCDT Decoding
63 03 94 A3 E8 FF 06 4B	HostRequestCode=0x3=ModeRequest HostServiceData=0x03=FlasherMode SecurityKey[0] SecurityKey[1] SecurityKey[2] SecurityKey[3] Dummy value, not necessary CRC	80 60 7F 1F F1 1F FF C9	ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) CRC
A0 00 00 00 00 00 0D 82		43 60 84 1F FD 20 01 0F	ProcessingStatus=b010=ResponsePending, RequestAck=0x3=ModeRequest ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) E2eCounter TripDC, CurrentCH1 TripAC, CurrentCH2
A0 00 00 00 00		83 60 81 00 00	ProcessingStatus=b100=PositiveResponse, RequestAck=0x3=ModeRequest ModuleState=b010=RcdActiveMode, ModuleData=b00000 (No fault) FirstFrameIndicator=1,

00 0E AC		00 00 4D	DataSequenceIndex=0x01=frame number 1 ServiceResponseData: None

1.6.1.5 ResetRequest

- Send OperationRequest frame with HostRequest set to "Reset request"
- Receive ServiceResponse frame

Note 1: the reset is performed just after the PositiveResponse frame is answered.

Note 2: To wait until the xCDT is ready: Wait for the max startup time duration described in Product datasheet, or in a more optimized way: perform an active wait by flooding with application frames until receiving a valid CRC and ModuleState=RcdActiveMode

1.6.1.5.1 Nominal example

The SPI exchange for a ResetRequest sent while the RCD is in ServiceMode can be as following:

Note: each line corresponds to a SPI exchange; the numbers are in hexadecimal format.

Master	RCD	Decoding
64 00 00 00 00 00 00 C3	80 60 D4 60 00 00 00 00	Master requests for ResetRequest=0x64 ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) CRC
A0 00 00 00 00 00 00 AD	44 60 D7 5F F5 5F FD 5C	RCD responds ProcessingStatus=b010=ResponsePending, RequestAck=0x4=ResetRequest ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) RcdE2eCounter TripDC, CurrentCH1 TripAC, CurrentCH2 CRC
A0 00 00 00 00 00	84 60 81 00 00 00	RCD responds ProcessingStatus=b100=PositiveResponse, RequestAck=0x4=ResetRequest ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) FirstFrameIndicator=1, DataSequenceIndex=1

00 AD	00 68	ServiceResponseData=[0x0000, 0x0000] CRC

1.6.1.6 Sw Product identification

- Send OperationRequest frame with HostRequest set to "Product identification, SW ID"
- Receive ServiceResponse frame as a long frame of 15 sub-frames (each sub-frame has 4B of payload):

ServiceResponseData	Size (Bytes)
SwIdentification: "B.D.R.C" format coded as 4 ASCII characters: Baseline, Delivery, Release, Correction	4
SwGitHashCode: short GIT tag of 7 ASCII characters, followed by "C" clean letter	8
SwSha256Code: 64 ASCII characters, representing the 32 Bytes SHA256 of application.	32
McuId: 16 bits word, representing the Device ID for the dsPIC33CK256MC506 Family • 0xA200: dsPIC33CK128MC102	2
Bootloader SwIdentification: Bootloader "B.D.R.C" format coded as 4 ASCII characters: Baseline, Delivery, Release, Correction	4
Bootloader SwGitHashCode: Bootloader short GIT tag of 7 ASCII characters, followed by "C" clean letter	8

1.6.1.6.1 Nominal example

To return the following Sw Product identification:

- SwIdentification 2.6.4.0
- GitHash 87e3608 C
- Sha256 94D2A42A989F8DF5FB297EABC4FB390C9658054E5AACC1C7B58281E6DE2DC190
- McuId 0xA200 (corresponds to dsPIC33CK128MC102)
- Fsw SwIdentification 2.2.2.0
- Fsw SwGitHashCode 81b2d83 C

the SPI exchange for a Sw Product identification request sent while the xCDT is in ServiceMode can be as following:

Note: each line corresponds to a SPI exchange; the numbers are in hexadecimal format.

Master	Master Decoding	xCDT	xCDT Decoding
61 00 00 00 00 00 00 1B	HostRequestCode=0x1=ProductIdentification HostServiceData=0x00=SW ID	80 60 DC 60 02 5F FC 5A	ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) CRC
A0 00 00 00 00 00 00 AD		41 60 E0 60 18 5F FD DA	ProcessingStatus=b110=ResponsePending, RequestAck=0x1=ProductIdentification ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) RcdE2eCounter TripDC, CurrentCH1 TripAC, CurrentCH2 CRC
A0 00 00 00 00 00 00 AD		81 60 8F 32 36 34 30 7D	ProcessingStatus=b100=PositiveResponse, RequestAck=0x1=ProductIdentification ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) FirstFrameIndicator=1, DataSequenceIndex=0x0F=frame number 15 ServiceResponseData=[0x32,0x36,0x34,0x30]="2.6.4.0"
...	...	...	...
A0 00 00 00 00 00 00 AD		81 60 01 64 38 33 43 75	DataSequenceIndex=frame number 1 ServiceResponseData=[0x64, 0x38, 0x33, 0x43] = d83C

1.6.1.7 Hw Product identification

- Send OperationRequest frame with HostRequest set to "Product identification, HW ID"
- Receive ServiceResponse frame as a long frame, of 52 frames containing 2 ASCII characters:

ServiceResponseData	Type	Size (Bytes)

PcbaProductionLog_Checksum	16 bits, Reserved	2
PcbaProductionLog_Size	16 bits, Reserved	2
PcbaProductionLog_Version	16 bits, Reserved	2
PcbaProductionLog_PcbaProductionDatecode	16 ASCII characters Format: PYYDDDCCHHMMSSJX	32
PcbaProductionLog_PcbaCLEM	18 ASCII characters Format: XX.XX.XX.XXX.X_VXX	36
PcbaProductionLog_Spare1	16 bits, Reserved	2
AssemblyProductionLog_Checksum	16 bits, Reserved	2
AssemblyProductionLog_Size	16 bits, Reserved	2
AssemblyProductionLog_Version	16 bits, Reserved	2
AssemblyProductionLog_SensorCLEM	14 ASCII characters Format: 90.W4.A2.200.0	28
AssemblyProductionLog_AssemblyProductionDatecode	16 ASCII characters	32
AssemblyProductionLog_AssemblyCustomerIdentificationNumber	32 ASCII characters	64
AssemblyProductionLog_Spare1	16 bits, Reserved	2

Note:

Total size is 208 Bytes, returned in 208 / 4 = 52 frames

An ASCII character is coded on a Word (2 Bytes). So each Service frame contains 4 Bytes of payload, corresponding to 2 ASCII characters.

1.6.1.7.1 Nominal example

To return the following Hw Product identification:

- PcbaProductionLog_Checksum: 0
- PcbaProductionLog_Size: 76
- PcbaProductionLog_Version: 2
- PcbaProductionLog_PcbaProductionDatecode: 9241459900565518
- PcbaProductionLog_PcbaCLEM: 93.52.63.801.0_V10
- Spare1: 0x0000
- AssemblyProductionLog_Checksum: 0
- AssemblyProductionLog_Size: 132
- AssemblyProductionLog_Version : 2
- AssemblyProductionLog_SensorCLEM: "90.W4.A2.200.0"
- AssemblyProductionLog_AssemblyProductionDatecode: "9241459900565517"

- AssemblyProductionLog_AssemblyCustomerIdentificationNumber: "ABCDEFGHIJKLMNOPQRSTUV0123456789"
- AssemblyProductionLog_Spare1: 0

the SPI exchange for a Hw Product identification request sent while the RCD is in ServiceMode can be as following:

Note: each line corresponds to a SPI exchange; the numbers are in hexadecimal format.

Master	Master Decoding	RCD	RCD Decoding
61 01 00 00 00 00 08 0C	HostRequestCode=0x1=ProductIdentification HostServiceData=0x01=HW ID	80 60 41 60 03 5F FF 06	ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) CRC
61 01 00 00 00 00 08 0C		41 60 45 60 03 60 00 CF	ProcessingStatus=b110=ResponsePending, RequestAck=0x1=ProductIdentification ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) RcdE2eCounter TripDC, CurrentCH1 TripAC, CurrentCH2 CRC
61 01 00 00 00 00 08 0C		81 60 B4 00 00 00 4C 82	ProcessingStatus=b100=PositiveResponse, RequestAck=0x1=ProductIdentification ModuleState=b011=ServiceMode, ModuleData=b00000 (No fault) FirstFrameIndicator=1, DataSequenceIndex=0x32=frame number 52 ServiceResponseData=[0x0000,0x004C]= 0, 76
61 01 00 00 00 00 08 0C		81 60 33 00 02 00 39 E4	FirstFrameIndicator=0, DataSequenceIndex=0x31=frame number 51 ServiceResponseData=[0x0002,0x0039]= 2, "9"

...	...	...	...
61 01 00 00 00 00 08 0C		81 60 01 00 39 00 00 90	FirstFrameIndicator=0, DataSequenceIndex=0x01=frame number 1 ServiceResponseData=[0x0039,0x0000]= "9", 0

1.6.1.8 PrimaryMeasurement

- Send OperationRequest frame with HostRequest set to Primary measurement readout
  - Receive ServiceResponse frame as a long frame (7 frames):

DataSequenceIndex	ServiceData[3] 1 Byte	ServiceData[4] 1 Byte	ServiceData[5] 1 Byte	ServiceData[6] 1 Byte
7	CurrentCH1 14bits unsigned		CurrentCH2 14bits unsigned	
6	MagOffsetCurrentPositive 16 bits signed (unit is 0.1 mA)		MagOffsetCurrentNegative 16 bits signed (unit is 0.1 mA)	
5	Bridge_CH1_Pwm1 16 bits unsigned (unit is 5 ns)		Bridge_CH1_Pwm2 16 bits unsigned (unit is 5 ns)	
4	Bridge_CH2_HalfPeriod1 16 bits unsigned (ADC bit unit) (For a NSF product: 0xFFFF)		Bridge_CH2_HalfPeriod2 16 bits unsigned (ADC bit unit) (For a NSF product: 0xFFFF)	
3	Vref2V5 16 bits unsigned (ADC bit unit)		Vcc5V 16 bits unsigned (ADC bit unit)	
2	McuTemperature 16 bits unsigned (ADC bit unit)		NtcTemperature 16 bits unsigned (ADC bit unit)	
1	E2eCounter 8 bits unsigned	Spare		

Description:

- Bridge_CH1_Pwm1, Bridge_CH1_Pwm2, Bridge_CH2_HalfPeriod1 and Bridge_CH2_HalfPeriod2 are Reserved values.
- Vref (V) = Vref2V5 * 3.3 / 4095
  - Note: Vref2V5 = 0x1000 means NotAvailable
  - This measure is not calibrated (Raw ADC value)
- Vcc (V) = Vcc5V * 2 * 3.3 / 4095
  - Note: Vcc5V = 0x1000 means NotAvailable
  - This measure is not calibrated (Raw ADC value)
- McuTemperature is a Reserved value

- NtcTemperature: To convert the PCB temperature to Celsius degrees, see the NTC Thermistor datasheet: https://www.tdk-electronics.tdk.com/web/designtool/ntc/. Use the table: NTC_B57232V5103F360_calcResult_from TDK.xlsx
  - Note: 0x1000 means NotAvailable
- E2eCounter has not to be considered

1.6.1.8.1 Nominal example

To return the following PrimaryMeasurement:

- CurrentCH1 -0.4mA
- CurrentCH2 0.0mA
- Bridge_CH1_Pwm1 4685
- Bridge_CH1_Pwm2 4676
- Bridge_CH2_HalfPeriod1 0
- Bridge_CH2_HalfPeriod2 0
- Vref 2.50 V
- Vcc 4.70 V
- Mcu temperature 947
- PCBA Temperature 1758 = 32.5 degC
- E2eCounter=0

the SPI exchange can be as following:

Note: each line corresponds to a SPI exchange; the numbers are in hexadecimal format.

Master	Master Decoding	RCD	RCD Decoding
6F 04 00 00 00 00 0D C1	HostRequestCode=0xF=PrimaryMeasurement HostServiceData=0x04 Dummy value, not necessary CRC	80 A0 CC 1F FB 20 00 91	ModuleState=b101=ReservedMode, ModuleData=b00000 (No fault) CRC
6F 04 00 00 00 00 0D C1	Operation request is repeated all along the sequence: this is not mandatory but it is a good practice.	4F A0 D0 20 00 20 01 77	ProcessingStatus=b010=ResponsePending, RequestAck=0xF=PrimaryMeasurement E2eCounter TripDC, CurrentCH1 TripAC, CurrentCH2
6F 04 00 00 00		8F A0 87 1F FC	ProcessingStatus=b100=PositiveResponse, RequestAck=0xF=PrimaryMeasurement FirstFrameIndicator=1, DataSequenceIndex=0x07=frame number 7

00 0D C1		20 00 88	ServiceResponseData: CurrentCH1=0x1FFC=-0.4 mA, CurrentCH2=0x2000=0 mA
6F 04 00 00 00 00 0D C1		8F A0 06 00 00 00 00 3A	DataSequenceIndex=0x06=frame number6 ServiceResponseData=[0x0,0x0,0x0,0x0]=Spare
6F 04 00 00 00 00 0D C1		8F A0 05 12 4D 12 44 12	DataSequenceIndex=0x05=frame number 5 ServiceResponseData: Bridge_CH1_Pwm1=0x124D=4685, Bridge_CH1_Pwm2=0x1244=4676
6F 04 00 00 00 00 0D C1		8F A0 04 00 00 00 00 29	DataSequenceIndex=0x04=frame number 4 ServiceResponseData: Bridge_CH2_HalfPeriod1=0, Bridge_CH2_HalfPeriod2=0
6F 04 00 00 00 00 0D C1		8F A0 03 0C 23 0B 68 73	DataSequenceIndex=0x03=frame number 3 ServiceResponseData: Vref=0xC23=2.50 V, Vcc=0xB68=4.70 V
6F 04 00 00 00 00 0D C1		8F A0 02 03 B3 06 DE A6	DataSequenceIndex=0x02=frame number 2 ServiceResponseData: Mcu temperature=0x3B3=947 ADC bit, PCBA Temperature=0x6DE=1758 ADC bit=32.5 degC
6F 04 00 00		8F A0 01 00	DataSequenceIndex=0x01=frame number 1

00 00 0D C1		00 00 00 CD	ServiceResponseData: E2eCounter=0

1.6.1.9 LowPowerModeRequest

- Send OperationRequest frame with HostRequest set to LowPowerMode
- Receive ServiceResponse frame

After having sent the LowPowerModeRequest, a reset is made and the product goes to sleep.

Wakeup procedure is as follows: Master shall send ApplicationFrames with a frequency of at least 1 KSps for about 200ms. (babbling-idiot/EMC disturbance protection)

1.6.1.10 ReadFaultContextRequest

- Send OperationRequest frame with HostRequest set to Read Fault Context
- Receive ServiceResponse frame as a long frame (13 frames):

DataSequenceIndex	ServiceData[3]	ServiceData[4]	ServiceData[5]	ServiceData[6]
13	FaultCode		ExtendedFaultCode	
12	ExtendedTrace[0]		ExtendedTrace[1]	
11	ExtendedTrace[2]		ExtendedTrace[3]	
10..1	Reserved data		Reserved data	

1.7 Network Layer

--

1.7.1 SPI mode

The xCDT sensor is acting as **Slave** on SPI bus.

Communication is **full-duplex**, initiated by Master. While the Master is sending a frame to xCDT, xCDT responds to the previous command received.

1.7.2 CRC-8 definition

The last byte in the 8 bytes SPI frame shall be the CRC computed on the 7 previous bytes.

The CRC shall comply with the following characteristics:

- Size: 8 bits
- Polynom: 0x97
- Initialization value: 0xFD
- No final XOR calculation on CRC value

Example of CRC calculation:

CRC calculated on [0xA0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00] is 0xAD

1.7.3 Master HostRequestFrame layout

Master shall send a HostRequestFrame accordingly to following Rx frame layout.

Byte0		Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
HostRequest		HostServiceData						CRC-8
HostCommand[3b]	HostRequestCode[5b]							

1.7.3.1 ApplicationRequest frame

HostCommand shall be set to b101 (ApplicationRequest)

HostRequestCode is not used, it shall be set to b00000 (NoHostRequest).

So HostRequest is set to 0xA0.

HostServiceData has one byte used: HostServiceData[1] corresponding to E2eInit. Other bytes can be set to zero. So HostServiceData is defined as [0x00 E2eInit 0x00 0x00 0x00 0x00].

To sum up, a typical ApplicationRequest frame is: [0xA0 0x00 E2eInit 0x00 0x00 0x00 0x00 Crc]

1.7.3.2 OperationRequest frame

HostCommand shall be set to b011 (OperationRequest).

HostRequestCode is set depending on the operation, as defined in the next table.

So HostRequest is in range 0x60..0x7F.

HostServiceData depends on HostRequest value. See next table:

HostRequest Code 4..0	HostRequest Byte0	HostServiceData Byte1	HostServiceData Byte2	HostServiceData Byte3	HostServiceData Byte4	HostServiceData Byte5	HostServiceData Byte6
b00000=0x00 Unsupported HostRequest Code	0x60	-	-	-	-	-	-
0x01 ProductIdentification	0x61	0x00 SW ID	-	-	-	-	
	0x61	0x01 HW ID	-	-	-	-	
Reserved	0x62	-	-	-	-	-	
0x03 ModeRequest	0x63	0x00 HardwareInitMode	E2eInit	-	-	-	
	0x63	0x01 LowPowerMode	-	-	-	-	
	0x63	0x02 Reserved mode	-	-	-	-	
	0x63	0x03 FlasherMode	Security Key, Flasher 0x94 0xA3 0xE8 0xFF				-
	0x63	0x04 ServiceMode	-	-	-	-	
0x04 ResetRequest	0x64	-	-	-	-	-	
0x05..0x8 Unsupported HostRequest Code	0x65..0x68	-	-	-	-	-	
0x0B..0x0D Reserved for LEM	0x69..0x6D	-	-	-	-	-	
0x0E Unsupported HostRequest Code	0x6E	-	-	-	-	-	

0x0F PrimaryMeasurement	0x6F	0x00	-	-	-	-
0x10 Unsupported HostRequest Code	0x70	-	-	-	-	-
0x11 Read Fault Context	0x71	-	-	-	-	-
0x12..0x1F Unsupported HostRequest Code	0x72..0x7F	-	-	-	-	-

Note: - (hyphen) means the value is ignored

1.7.4 xCDT ResponseFrame layout

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
MasterAck	Module	ResponseData					CRC-8

1.7.4.1 ApplicationResponse frame

xCDT responds to the ApplicationRequest frame accordingly to the following layout.

Byte0		Byte1		Byte2	Byte3	Byte4	Byte5		Byte6	Byte7
MasterAck		Module		E2eCounter	TrippingData					CRC-8
7..5	4..0	7..5	4..0		7..6	5..0	7..0	7..6	5..0	7..0
ProcessingStatus	RequestAck	ModuleState	ModuleData		TripDC	CurrentCH1		TripAC	CurrentCH2	

1.7.4.1.1 ProcessingStatus

ProcessingStatus =MasterAck[7..5]	Name	Description

0 = b000	IncorrectMessageLengthOrInvalidFormat	The master is sending a bad header frame OR The master is sending a frame with incorrect size
1 = b001	InvalidChecksum	Master CRC-8 is incorrect
2 = b010	ResponsePending	RCD is processing the command
3 = b011	RequestNotSupported	Unknown command
4 = b100	MessageReceivedApplication or PositiveResponse	Master frame received is correct
5 = b101	InvalidE2eInit or SecurityAccessDenied	- Invalid E2E counter initialization value from Master - Incorrect password
6 = b110	ConditionsNotCorrect/requestSequenceError	Mode change request from incorrect mode
7 = b111	Spare	-

1.7.4.1.2 RequestAck

Equal to the HostRequestCode received

1.7.4.1.3 ModuleState

Indicate the current product state

ModuleState = Module[7..5]	Name
0 = b000	Spare, shall be treated as error by Master
1 = b001	HardwareInitMode
2 = b010	RcdActiveMode
3 = b011	ServiceMode
4 = b100	Reserved
5 = b101	Reserved
6 = b110	FallbackMode
7 = b111	IntegrityFailMode

1.7.4.1.4 ModuleData

ModuleData meaning depends on ModuleState value.

ModuleState = Module[7..5]	ModuleData = Module[4..0]				
	bit 4	bit 3	bit 2	bit 1	bit 0
RcdActiveMode = 2	*Temperature thresholds* 0: Temperature <= 56.5°C 1: Temperature > 56.5°C 2: Temperature > 88.5°C 3: Temperature > 104.5°C 4: Temperature > 112.5°C 5: Temperature > 116.5°C 6: Temperature > 118.5°C 7: Temperature > 119.5°C			*This state is reached by coming from* 0: Startup (after a power-on or a reset) 1: SPI Request (HardwareInitModeRequest has been requested by master.) 2: OvercurentPrefail 3: FallbackMode (another cause than OvercurentPrefail)	
FallbackMode = 6 OR ServiceMode = 3	0: No fault *Power supply fault* 1: Over voltage fault 2: Under voltage fault *Operating temperature fault* 3: Over temperature fault 4: Under temperature fault *Primary current/Fluxgate fault* 5: Overcurrent PreFailed, I > 600..700 mA during glitch filter time (500 us) 6: Overcurrent Confirmed fault, I > 600..700 mA after glitch filter time (500 us)			Reserved	
HardwareInitMode = 1	*Initialisation steps* 0: Spare 1: Hardware initialization 2: Control magnetization 3: LowPower wakeup confirmation 4: Hardware Initialization Lite (after an Overcurrent Prefailed condition)				
IntegrityFailMode = 7	Integrity fault code to be communicated to LEM support				

1.7.4.1.5 E2eCounter

The E2eCounter is coded on 1 Byte.

E2eCounter	Signification	Description
0	Initialization	Value after xCDT reset
1..254	Normal	Increments up to 254 then loops back to 1
255	Overflow	Master has not sent any ApplicationRequest frame since the E2E counter started from value 1

1.7.4.1.6 TripDC

Trip signal for DC current is coded on 2 bits.

TripDC value	Tripping status	Signification	Description
0	Inactive	Inactive	No DC leakage current is detected. Note: sensor is in RcdActiveMode
1	Active Open relays	Active	A fault is detected due to DC leakage current. Note: sensor is in RcdActiveMode
2	Active Open relays	NotAvailable	Sensor is initializing Note: sensor is in HardwareInitMode, ServiceMode or FactoryMode
3	Active Open relays	Error	Sensor is in Error state Conditions to occur: • A ChannelCorrelation error is occurring while the sensor is in RcdActiveMode • OR the sensor is in IntegrityFailMode or FallbackMode (except during Overcurrent PreFailed status)

1.7.4.1.7 CurrentCH1

CurrentCH1 is coded on 14 bits in big endian format, with a 100 uA resolution and a +8192 offset. It corresponds to an instantaneous value (not RMS)

CurrentCH1_mA = (CurrentCH1 - 0x2000) * 0.1

CurrentCH1 value (LSB format)	Value if decoded as a current	Signification	Description
0x3FFF	819.1 mA	Not available	during bridge initialisation
0x3FFE	819.0 mA	Error	If xCDT is in FallbackMode due to a Voltage or Temperature fault If xCDT is in IntegrityFailMode
0x3FFD	818.9 mA	Bridge saturation	when |leakage current| > [500..700] mA

1.7.4.1.8 TripAC

Trip signal for AC current is coded on 2 bits.

TripAC value	Tripping status	Signification	Description
0	Inactive	Inactive	No AC leakage current is detected. Note: sensor is in RcdActiveMode
1	Active Open relays	Active	A fault is detected due to AC leakage current Note: sensor is in RcdActiveMode
2	Active Open relays	NotAvailable	Sensor is initializing Note: sensor is in HardwareInitMode, ServiceMode or FactoryMode
3	Active Open relays	Error	Sensor is in Error state Conditions to occur: • A ChannelCorrelation error is occurring while the sensor is in RcdActiveMode • OR the sensor is in IntegrityFailMode or FallbackMode (except during Overcurrent PreFailed status)

1.7.4.1.9 CurrentCH2

CurrentCH2 is coded on 14 bits in big endian format, with a 100 uA resolution and a +8192 offset. It corresponds to an instantaneous value (not RMS)

CurrentCH2_mA = (CurrentCH2 - 0x2000) * 0.1

CurrentCH2 value (LSB format)	Value if decoded as a current	Signification	Description
0x3FFF	819.1 mA	Not available	during bridge initialisation or in case of non-SF product
0x3FFE	819.0 mA	Error	If xCDT is in FallbackMode due to a Voltage or Temperature fault If xCDT is in IntegrityFailMode
0x3FFD	818.9 mA	Overcurrent	• Raise fault if |CH1 digital leakage current| > 300 mA • Release fault if |CH1 digital leakage current| < 280 mA

1.7.4.2 ServiceResponse frame

xCDT responds to the OperationRequest frame accordingly to the following layout.

Byte 0		**Byte 1**		**Byte 2**		**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**
MasterAck		Module		FirstFrameIndicator / DataSequenceIndex		ServiceResponseData				CRC-8
ProcessingStatus 7..5	RequestAck 4..0	ModuleState 7..5	ModuleData 4..0	FirstFrameIndicator 7	DataSequenceIndex 6..0	Response [Index +4]	Response [Index +3]	Response [Index +2]	Response [Index +1]	

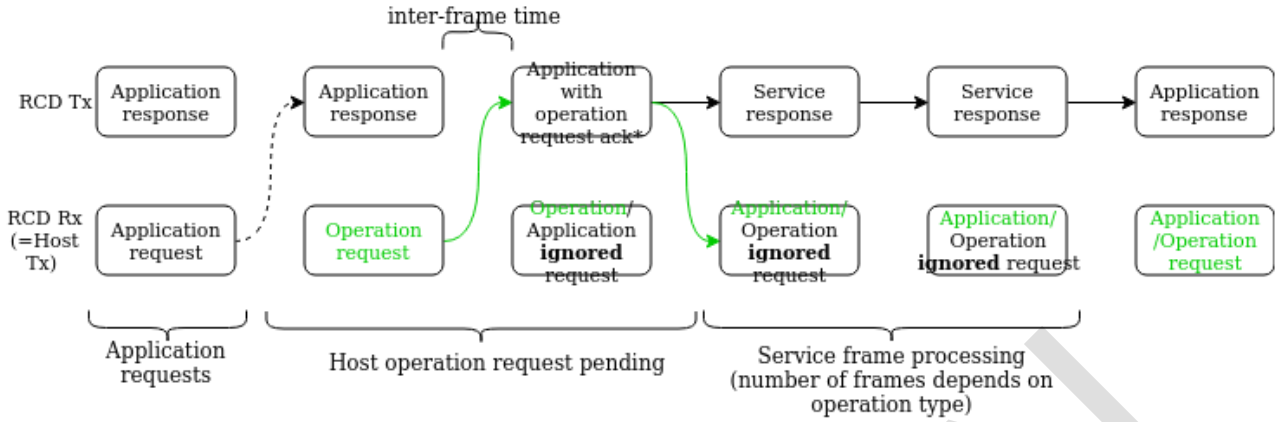
Notes:

- DataSequenceIndex is decrementing, see Long frame processing.
- Index = (DataSequenceIndex - 1) * 4

1.7.4.2.1 ServiceResponse frame processing

Product provides on each host communication request:

- ApplicationResponseFrame, if no OperationRequest has been received in previous frame
- ApplicationResponseFrame with RequestAck status equal to the HostRequestCode received, if OperationRequest has been received
- ServiceResponseFrame answering received Service/Operation HostRequest, until the Operation is completed (e.g. multi-frame response)



*Acknowledge frame is provided in next Tx frame, if the communication period allows the Tx frame update (inter-frame time >30µs), otherwise the "request ack" will be provided in next Tx frame.

Green marked elements indicates the nominal/recommended Host operation behavior.

Note: If the Host stops sending frames for more than 2.5 ms while a multi-frame response is ongoing, then the RCD aborts the current service response and go back to normal appFrames.

1.7.4.2.1.1 Long frame processing

In case the answer is longer than 4 bytes, the long frame processing is used.

In order to support the data chunk decomposition (long frame protocol), Product provides the responses in following way:

- Bit7 is reserved for the first frame indication
- DataSequenceIndex byte indicates the "data index":
  - 1: simple frame, all data is contained in the response
  - 2..127: long frame data, value indicating the position of the data inside of the long frame
    - Transmission of long frame shall be started by most significative value, rationale: allows to sender/receiver to identify the size of the long frame from the first "frame" reception.

Example for a 5 frames chunk, allowing 5 x 4 = 20 Bytes to be answered:

DataSequenceIndex, frame1	**DataSequenceIndex, frame2**	**DataSequenceIndex, frame3**	**DataSequenceIndex, frame4**	**DataSequenceIndex, frame5**
0x85 FirstFrameIndicator (bit 7) is raised	0x04	0x03	0x02	0x01

1.8 Physical Layer

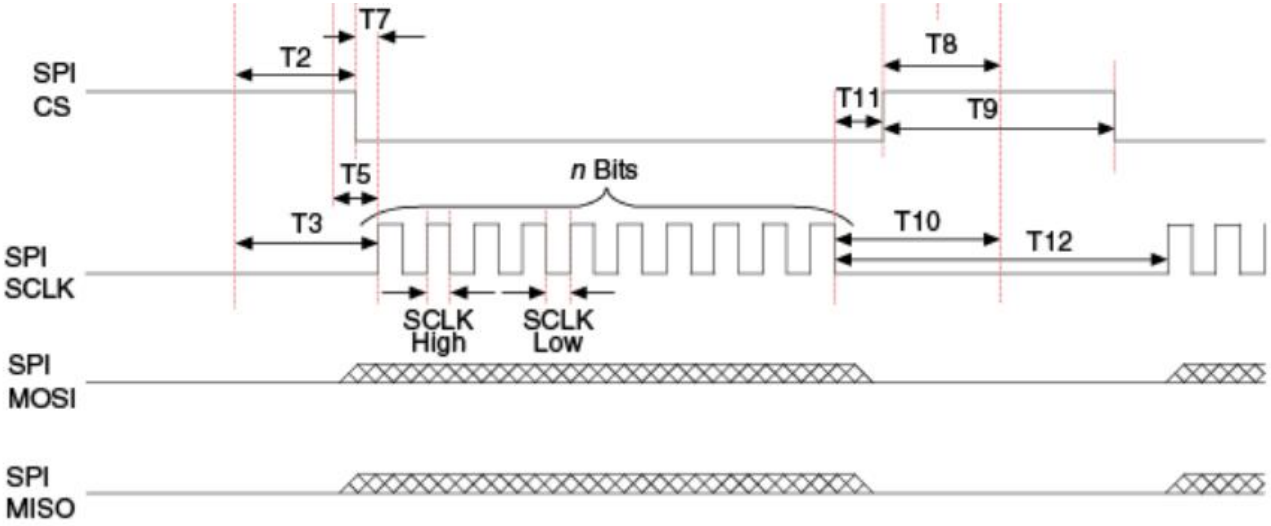
--

1.8.1 HW characteristics

- SPI signal levels are **3.3 V**
- SCLK is driven by Master
  - SPI Clock shall be **1 MHz +/- 10 KHz**
  - SPI Clock polarity shall be: **Clock is low in the idle state**
  - SPI Clock phase shall be: **Data is centered on the second edge of the clock period**
- /SS: SPI Slave Select is driven by Master
  - /SS signal shall go at state low (ground) during the SPI frames: /SS signal is active at low state
  - /SS signal shall go at state high between the SPI frames: /SS signal is inactive at high state
- MOSI is driven by Master
- MISO is driven by xCDT when /SS is active

1.8.2 Frame characteristics

- SPI requests shall be **8 bytes** long. (=SPI_frame_size)
- T7 > 4 us : The delay between the SS active low signal and the start of SPI clock must be **at least 4 microseconds** long. (=delay_after_SS_active).
- The maximum SPI sample rate is **1 KSps**. (for Safety usage, it shall be 1 KSps +/- 10%). So, the start of SPI requests shall be time-spaced by at least 1 millisecond (=min_SPI_time_start).
- T9: The minimum interframe duration is min_SPI_time_start - delay_after_SS_active - (SPI_frame_size * 8 * SPI_clock_period) = 1000 - 4 - (8 * 8 * 1) = 932 microseconds
- No dead time between two bytes shall be inserted
- SCLK High = SCLK Low = 1 us
- n = 64 Bits
- T11 > 650 ns: Enable lag time (CLK to rising edge of SS). Tested with 650 ns, shorter values may work



1.9 Troubleshooting

--

1.9.1 SPI clock configuration

The SPI clock configuration shall be configured as described in HW characteristics:

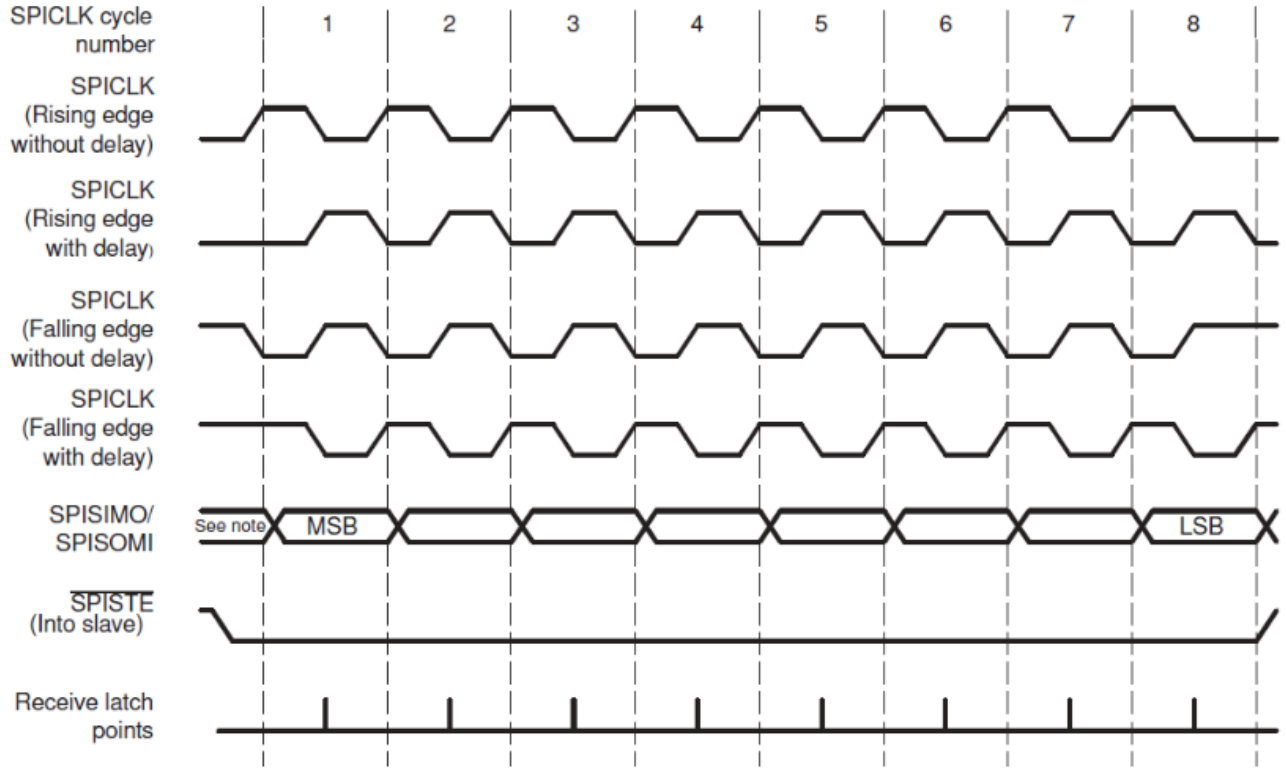
- SPI Clock polarity shall be: Clock is low in the idle state
- SPI Clock phase shall be: Data is centered on the second edge of the clock period

Usually, it corresponds to SPI Mode 1 with CPOL=0 and CPHA=1. This is the case for STM32 processors, FTDI UMFT SPI/USB adapters and NI SPI/USB adapters.

But for TI TMS320 processors, it corresponds to Mode 0 Polarity 0, Phase 0, "Rising edge without delay", as described in next schematic:

SPICLK Scheme	CLKPOLARITY	CLK_PHASE(1)
Rising edge without delay	0	0
Rising edge with delay	0	1
Falling edge without delay	1	0
Falling edge with delay	1	1

(1) The description of CLK_PHASE and CLKPOLARITY differs between manufacturers. For proper operation, select the desired waveform to determine the clock phase and clock polarity settings.



Note: Previous data bit

1.9.2 SPI CRC optimization

The SPI HW block of some STM32 processors provides the feature to check the validity of the received CRC without interrupting the CPU. Thanks to the implementation chosen, it should be possible to use this feature (not yet tested at LEM) because:

- CRC is located in the last byte of the SPI frame
- no final XOR operation is required

1.10 Annex

--

1.10.1 ApplicationRequest frame generation

Using Python language

```
import crcmod

crc_func = crcmod.mkCrcFun(0x197, rev=False, initCrc=0xFD, xorOut=0x00)

frame = [0xA0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]

crc = crc_func(bytearray(frame))
frame.append(crc)

print(frame)
# ApplicationRequest frame to send on SPI bus: [0xA0, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xAD]
```

1.10.2 ApplicationResponse frame decoding

Using Python language:

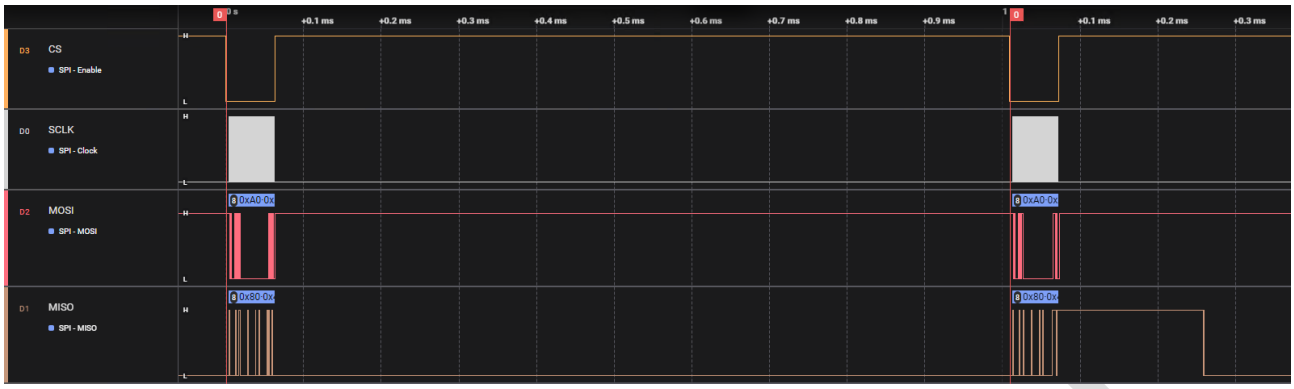
```
# Considering a frame received from SPI bus

ProcessingStatus    = frame[0] >> 5
HostRequest         = frame[0] & 0x1F

ModuleState         = frame[1] >> 5
ModuleData          = frame[1] & 0x1F

RcdE2eCounter       = frame[2]

TripDC              = frame[3] >> 6
rawCH1              = (frame[3] & 0x3F) << 8 | frame[4]
CurrentCH1          = (rawCH1 - 0x2000) * 0.1

TripAC              = frame[5] >> 6
rawCH2              = (frame[5] & 0x3F) << 8 | frame[6]
CurrentCH2          = (rawCH2 - 0x2000) * 0.1

CRC                 = frame[7]
```

1.10.3 Communication exchange

Here's a capture of SPI Master / xCDT communication at 1 KSps rate.

Master is a STM32H7A3ZI-Q board (Nucleo development board). The trace is recorded using a Saleae logic analyzer.

2 SPI frames sent every 1 ms (1 KSps):



Zoom in to show the 4 us delay between /SS and the start of SCLK: