BD940/BD940-INS/BX940 and BD99x

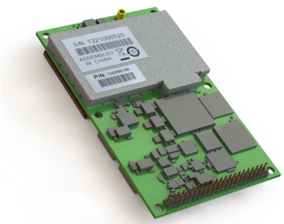
GNSS and Inertial Receiver

INTEGRATOR GUIDE

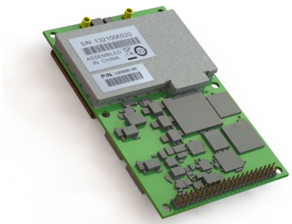
BD940

BD940-INS

BX940



BD990



BD992/BD992_INS

BX992

Version 5.32
Revision A
May 2018

Corporate Office

Trimble Inc.
Integrated Technologies
510 DeGuigne Drive
Sunnyvale, CA 94085
USA
www.trimble.com/gnss-inertial
Email: GNSSOEMSupport@trimble.com

Legal Notices

Support for Galileo is developed under a license of the European Union and the European Space Agency (BD910/BD920/BD930/BD935/BD940/BD970/BD982/BX935/BX982).

Release Notice

This is the May 2018 release (Revision A) of the BD940/BD99x Integrator Guide. It applies to version 5.32 of the receiver firmware.

LIMITED WARRANTY TERMS AND CONDITIONS

Product Limited Warranty

Subject to the following terms and conditions, Trimble Navigation Limited ("Trimble") warrants that for a period of one (1) year from date of purchase unless otherwise specified, this Trimble product (the "Product") will substantially conform to Trimble's publicly available specifications for the Product and that the hardware and any storage media components of the Product will be substantially free from defects in materials and workmanship.

Product Software

Product software, whether built into hardware circuitry as firmware, provided as a standalone computer software product, embedded in flash memory, or stored on magnetic or other media, is licensed solely for use with or as an integral part of the Product and is not sold. If accompanied by a separate end user license agreement ("EULA"), use of any such software will be subject to the terms of such end user license agreement (including any differing limited warranty terms, exclusions, and limitations), which shall control over the terms and conditions set forth in this limited warranty.

Software Fixes

During the limited warranty period you will be entitled to receive such Fixes to the Product software that Trimble releases and makes commercially available and for which it does not charge separately, subject to the procedures for delivery to purchasers of Trimble products generally. If you have purchased the Product from an authorized Trimble dealer rather than from Trimble directly, Trimble may, at its option, forward the software Fix to the Trimble dealer for final distribution to you. Minor Updates, Major Upgrades, new products, or substantially new software releases, as identified by Trimble, are expressly excluded from this update process and limited warranty. Receipt of software Fixes or other enhancements shall not serve to extend the limited warranty period.

For purposes of this warranty the following definitions shall apply: (1) "Fix(es)" means an error correction or other update created to fix a previous software version that does not substantially conform to its Trimble specifications; (2) "Minor Update" occurs when enhancements are made to current features in a software program; and (3) "Major Upgrade" occurs when significant new features are added to software, or when a new product containing new features replaces the further development of a current product line. Trimble reserves the right to determine, in its sole discretion, what constitutes a Fix, Minor Update, or Major Upgrade.

Warranty Remedies

If the Trimble Product fails during the warranty period for reasons covered by this limited warranty and you notify Trimble of such failure during the warranty period, Trimble will repair OR replace the nonconforming Product with new, equivalent to new, or reconditioned parts or Product, OR refund the Product purchase price paid by you, at Trimble's option, upon your return of the Product in accordance with Trimble's product return procedures then in effect.

How to Obtain Warranty Service

To obtain warranty service for the Product, please contact your local Trimble authorized dealer. Alternatively, you may contact Trimble to request warranty service by e-mailing your request to GNSSOEMSupport@trimble.com. Please be prepared to provide:

– your name, address, and telephone numbers

– proof of purchase

– a copy of this Trimble warranty

– a description of the nonconforming Product including the model number

– an explanation of the problem

The customer service representative may need additional information from you depending on the nature of the problem.

Warranty Exclusions or Disclaimer

This Product limited warranty shall only apply in the event and to the extent that (a) the Product is properly and correctly installed, configured, interfaced, maintained, stored, and operated in accordance with Trimble's applicable operator's manual and specifications, and; (b) the Product is not modified or misused. This Product limited warranty shall not apply to, and Trimble shall not be responsible for, defects or performance problems resulting from (i) the combination or utilization of the Product with hardware or software products, information, data, systems, interfaces, or devices not made, supplied, or specified by Trimble; (ii) the operation of the Product under any specification other than, or in addition to, Trimble's standard specifications for its products; (iii) the unauthorized installation, modification, or use of the Product; (iv) damage caused by: accident, lightning or other electrical discharge, fresh or salt water immersion or spray (outside of Product specifications); or exposure to environmental conditions for which the Product is not intended; (v) normal wear and tear on consumable parts (e.g., batteries); or (vi) cosmetic damage. Trimble does not warrant or guarantee the results obtained through the use of the Product, or that software components will operate error free.

NOTICE REGARDING PRODUCTS EQUIPPED WITH TECHNOLOGY CAPABLE OF TRACKING SATELLITE SIGNALS FROM SATELLITE BASED AUGMENTATION SYSTEMS (SBAS) (WAAS/EGNOS, AND MSAS), OMNISTAR, GPS, MODERNIZED GPS OR GLONASS SATELLITES, OR FROM IALA BEACON SOURCES: *TRIMBLE IS NOT RESPONSIBLE FOR THE OPERATION OR FAILURE OF OPERATION OF ANY SATELLITE BASED POSITIONING SYSTEM OR THE AVAILABILITY OF ANY SATELLITE BASED POSITIONING SIGNALS.*

THE FOREGOING LIMITED WARRANTY TERMS STATE TRIMBLE’S ENTIRE LIABILITY, AND YOUR EXCLUSIVE REMEDIES, RELATING TO THE TRIMBLE PRODUCT. EXCEPT AS OTHERWISE EXPRESSLY PROVIDED HEREIN, THE PRODUCT, AND ACCOMPANYING DOCUMENTATION AND MATERIALS ARE PROVIDED “AS-IS” AND WITHOUT EXPRESS OR IMPLIED WARRANTY OF ANY KIND, BY EITHER TRIMBLE OR ANYONE WHO HAS BEEN INVOLVED IN ITS CREATION, PRODUCTION, INSTALLATION, OR DISTRIBUTION, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT. THE STATED EXPRESS WARRANTIES ARE IN LIEU OF ALL OBLIGATIONS OR LIABILITIES ON THE PART OF TRIMBLE ARISING OUT OF, OR IN CONNECTION WITH, ANY PRODUCT. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW LIMITATIONS ON DURATION OR THE EXCLUSION OF AN IMPLIED WARRANTY, THE ABOVE LIMITATION MAY NOT APPLY OR FULLY APPLY TO YOU.

Limitation of Liability

TRIMBLE'S ENTIRE LIABILITY UNDER ANY PROVISION HEREIN SHALL BE LIMITED TO THE AMOUNT PAID BY YOU FOR THE PRODUCT. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL TRIMBLE OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGE WHATSOEVER UNDER ANY CIRCUMSTANCE OR LEGAL THEORY RELATING IN ANYWAY TO THE PRODUCTS, SOFTWARE AND ACCOMPANYING DOCUMENTATION AND MATERIALS, (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF DATA, OR ANY OTHER PECUNIARY LOSS), REGARDLESS OF WHETHER TRIMBLE HAS BEEN ADVISED OF THE POSSIBILITY OF ANY SUCH LOSS AND REGARDLESS OF THE COURSE OF DEALING WHICH DEVELOPS OR HAS DEVELOPED BETWEEN YOU AND TRIMBLE. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY OR FULLY APPLY TO YOU.

PLEASE NOTE: THE ABOVE TRIMBLE LIMITED WARRANTY PROVISIONS WILL NOT APPLY TO PRODUCTS PURCHASED IN THOSE JURISDICTIONS (E.G., MEMBER STATES OF THE EUROPEAN ECONOMIC AREA) IN WHICH PRODUCT WARRANTIES ARE THE RESPONSIBILITY OF THE LOCAL TRIMBLE AUTHORIZED DEALER FROM WHOM THE PRODUCTS ARE ACQUIRED. IN SUCH A CASE, PLEASE CONTACT YOUR LOCAL TRIMBLE AUTHORIZED DEALER FOR APPLICABLE WARRANTY INFORMATION.

Official Language

THE OFFICIAL LANGUAGE OF THESE TERMS AND CONDITIONS IS ENGLISH. IN THE EVENT OF A CONFLICT BETWEEN ENGLISH AND OTHER LANGUAGE VERSIONS, THE ENGLISH LANGUAGE SHALL CONTROL.

COCOM limits

This notice applies to the BD910, BD920, BD920-W, BD920-W3G, BD930, BD930-UHF, BD935-INS, BD960, BD970, BD982, BX960, BX960-2, and BX982 receivers.

The U.S. Department of Commerce requires that all exportable GPS products contain performance limitations so that they cannot be used in a manner that could threaten the security of the United States. The following limitations are implemented on this product:
– Immediate access to satellite measurements and navigation results is disabled when the receiver velocity is computed to be greater than 1,000 knots, or its altitude is computed to be above 18,000 meters. The receiver GPS subsystem resets until the COCOM situation clears. As a result, all logging and stream configurations stop until the GPS subsystem is cleared.

Restriction of Use of Certain Hazardous Substances in Electrical and Electronic Equipment (RoHS)

Trimble products in this guide comply in all material respects with DIRECTIVE 2002/95/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 January 2003 on the restriction of the use of certain hazardous substances in electrical and electronic equipment (RoHS Directive) and Amendment 2005/618/EC filed under C(2005) 3143, with exemptions for lead in solder pursuant to Paragraph 7 of the Annex to the RoHS Directive applied.

Waste Electrical and Electronic Equipment (WEEE)

For product recycling instructions and more information, please go to www.trimble.com/Corporate/Environmental_Compliance.aspx.

Recycling in Europe: To recycle Trimble WEEE (Waste Electrical and Electronic Equipment, products that run on electrical power.), Call +31 497 53 24 30, and ask for the "WEEE Associate". Or, mail a request for recycling instructions to:

Trimble Europe BV

c/o Menlo Worldwide Logistics

Meerheide 45

5521 DZ Eersel, NL

Contents

1

Power Requirements and Circuitry

BD940/BD940-INS

Power input

The unit, excluding the antenna operates at 3.3 V +5%/-3%. The 3.3 V should be able to supply 1.8 A of worst-case surge current. Full-load power consumption including antenna is 2.7 W.

Over-voltage protection

The absolute maximum voltage is 3.6 V.

Under-voltage protection

The absolute minimum voltage is 3.2 V below nominal.

Reverse voltage protection

The unit is protected down to -3.6 V.

Power On reset (PORESET)

There is a PORESET controller that monitors the power rails. The PORESET signal will be low if Maxwell Core, CPU Core, and CPU 1.8 V are not correct. The signal will stay low for 200 mSec after all of these rails are good. If any of the rails drop, PORESET will be toggled. PORESET is connected to a hardware reset on the CPU.

Antenna power out

Power output specification

The antenna DC power is supplied directly from Pin 3 on the multi-pin Interface Connector J100. The antenna output is rated to 10 V and can source a maximum of 150 mAmps.

Short-circuit protection

The unit does not have any over-current/short circuit protection related to the Antenna bias. Short circuits may cause damage to the Antenna port bias filtering components if the sourcing supply is not current limited to less than 150 mAmps.

Power consumption

This section provides details on power consumption for the BD940 module when configured to different operating modes. The testing environment is considered as ideal and therefore these numbers are for reference purposes only.

NOTE – It is important to consider the following caveats when using these numbers for integrating the BD940 into a larger system:

1. *Testing was done by placing the module on the Trimble BD940 evaluation board.*
2. *Voltage Input – 12 V DC to BD940 evaluation board.*
3. *Firmware Release – v5.33.*
4. *Antenna – Zephyr II Geo with power LNA–*
5. *Ethernet – Enabled with active embedded web interface.*
6. *RS-232 – Port speed 230,400 bps.*
7. *Output Protocol – NMEA @ 20 Hz (GGA, GST, and GSV).*
8. *RTK Navigation.*
9. *sCMRx Over Ethernet.*
10. *Ambient Temp: 25 °C.*

Legend Test Profile

- Test 1 L1 L2 RTK @ 20 Hz over RS-232 = 2.32 W Min and 2.40 W Max
- Test 2 L1 L2 G1 G2 RTK @ 20 Hz over RS-232 = 2.36 W and 2.47 W Max
- Test 3 L1 L2 G1 G2 SBAS RTK @ 20 Hz over RS-232 = 2.36 W Min and 2.47 W Max
- Test 4 L1 L2 G1 G2 SBAS L5 RTK @ 20 Hz over RS-232 = 2.55 W Min and 2.73 W Max
- Test 5 L1 L2 G1 G2 SBAS L5 Galileo RTK @ 20 Hz over RS-232 = 2.60 W Min and 2.76 W Max
- Test 6 L1 L2 G1 G2 SBAS L5 Galileo RTX over L-Band 20 Hz over RS-232 = 2.77 W Min and 2.95 W Max
- Test 7 L1 L2 G1 G2 SBAS L5 Galileo B1 B2 RTK @ 20 Hz over RS-232 = 2.65 W Min and 3.07 W Max

BD990/BD992/BD992-INS

Power supply

The unit operates with a nominal input of 3.3 V +5%/-3%. The external 3.3 V supply should be able to supply 1.8 A of worst-case surge current. Worst case full load power consumption including antenna is 5.0 W. (Note: Worst case was tested with all features, including RF bands, LEDs, enabled, at +85 °C). There are multiple power rails in the system. Voltage rails 1.2 V, 1 V, 1.8 V, 2.35 V, 5.7 V, or 7.6 V (antenna outputs) are provides by switching supplies. 3.1 V, 3.0 V, and 1.95 V use LDOs to achieve low noise voltage rails.

Power protection

The 3.3 V input is monitored by an LTC2912 for over and under voltage conditions. (Basically a window comparator). If the voltage exceeds 3.64 V or is under 3.01 V the IC turns off the gate of a MOSET to disconnect the input voltage to the system. Limited protection above 3.64 V is offered by a varistor which has a clamping voltage of 5.5 V.

Antenna power out

Each antenna connector can supply DC power independently. Each output is supplied by a dedicated boost regulator. The primary antenna regulator can switch voltage between 5.7 V to 7.6 V by using a GPIO to change the feedback and can source a maximum of 150 mAmps. Switching is done to select narrow vs wideband filtering for MSS jam-immunity in capable Trimble antennas. The output antenna features a constant 5.7 V output. Each antenna has a PTC with a hold current of 200 mA which limits the output current and provides short circuit protection.

Power switch and reset

Power switch

The integrator may choose to power on or power off the unit. If a 3.3 V level signal is applied to pin 3, Power_Off pin, the unit will disconnect VCC. The system integrator must ensure that other TTL level pins remain unpowered when Power_Off is asserted. Powering TTL-level pins while the unit is powered off will cause excessive leakage current to be sinked by the unit.

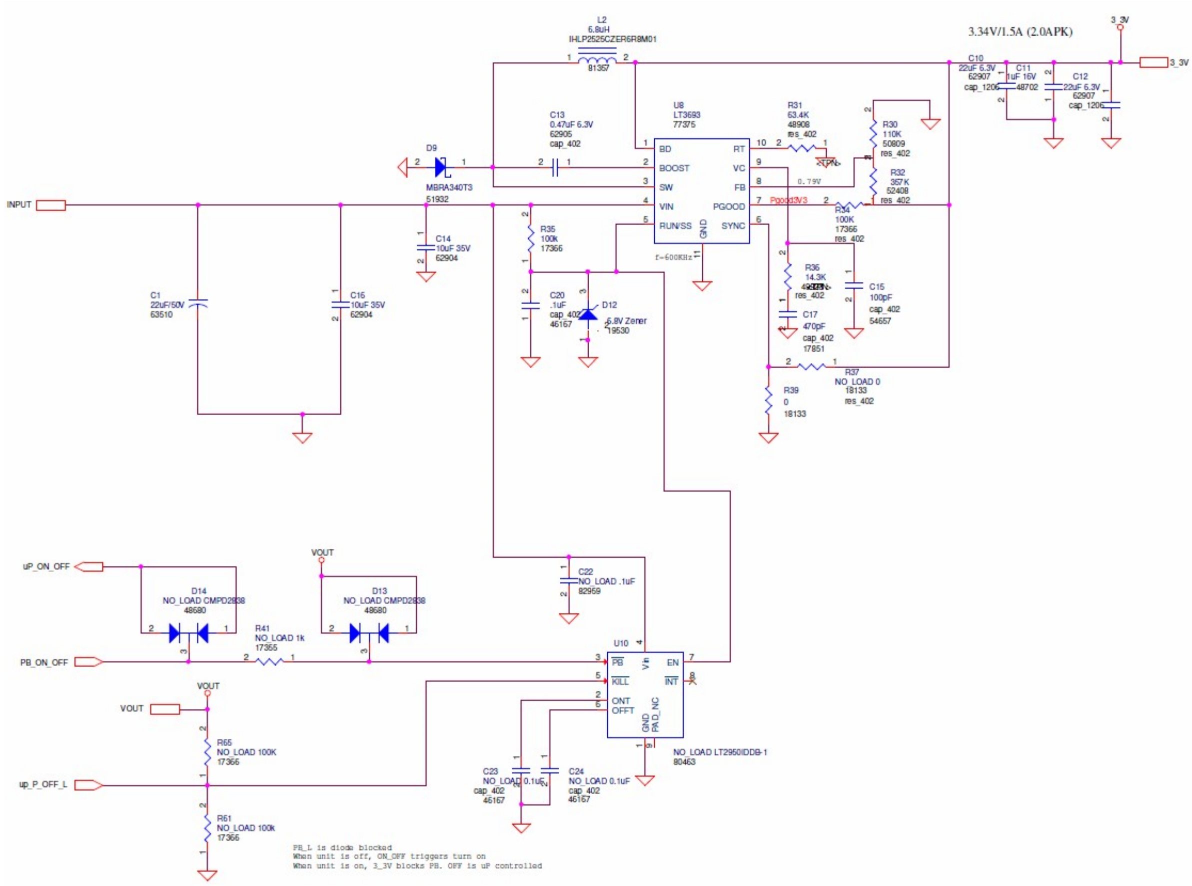
The integrator may choose to always have the unit powered on. This is accomplished by leaving the Power_Off pin floating or grounded.

Reset switch

Driving Reset_IN_L, Pin 12, low will cause the unit to reset. The unit will remain reset at least 140 mS after the Reset_In_L is deasserted. The unit remains powered while in reset.

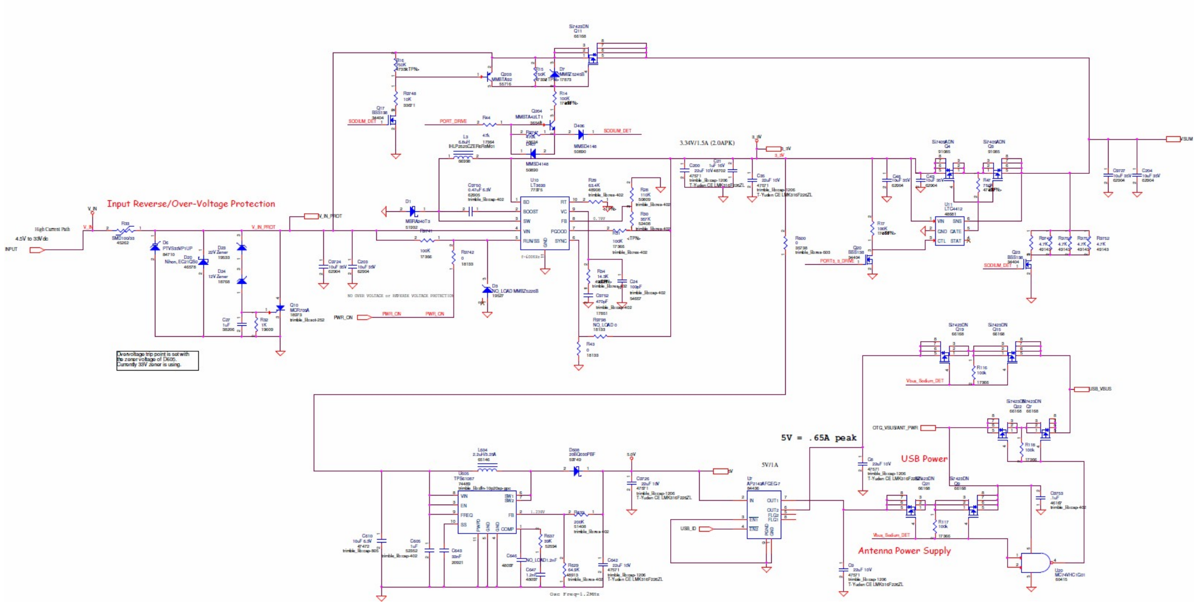
BD940 evaluation board

Power input circuit



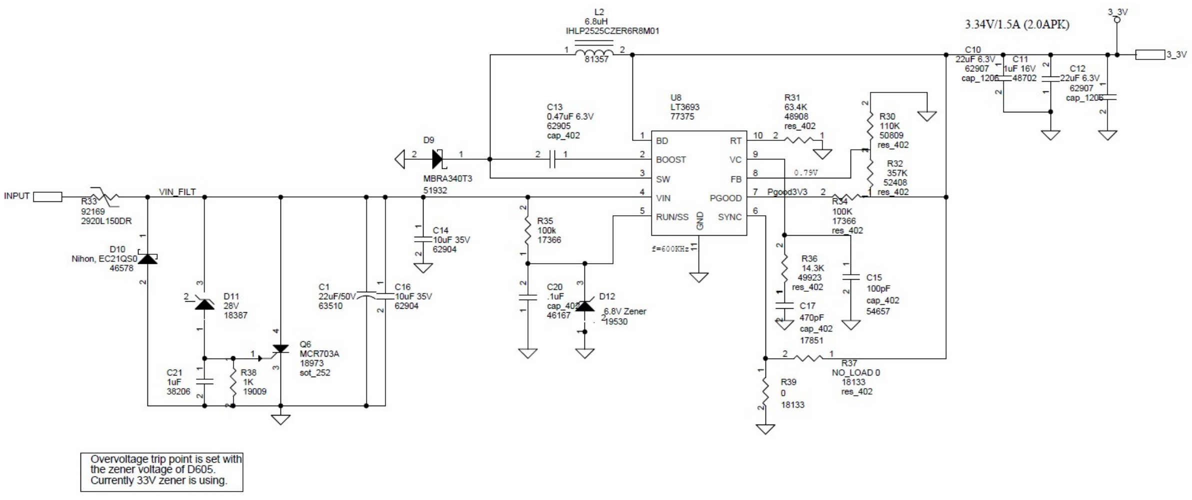
BD940-INS evaluation board

Power supply circuit



BD990/BD992/BD992-INS evaluation board

Power supply circuit



Antenna power output

Power output specification

The antenna supplies 100 mA at 5 V.

Short-circuit protection

The unit has an over-current / short circuit protection. Short circuits may cause the unit to reset.

2

Ethernet

The receiver contains the Ethernet MAC and PHY, but requires external magnetics. The PHY layer is based on the Micrel KSZ8041NLI it is set to default to 100 Mbps, full duplex with auto-negotiation enabled.

Since the Ethernet functionality will typically increase the receiver power consumption by approximately 10%, the receiver shuts down the Ethernet controller if no Ethernet devices are connected within two minutes.

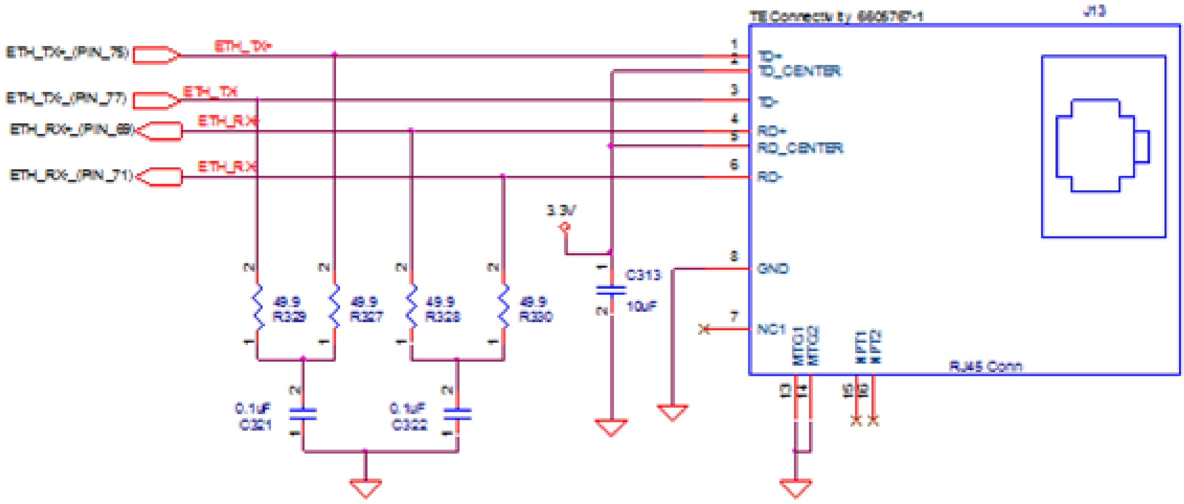
Isolation transformer selection

Parameters	Value	Test condition
Turns Ratio	1CT:1CT	
Open-circuit inductance (min.)	350 uH	100 mV, 100 kHz, 8 mA
Leakage inductance (max.)	0.4 uH	1 MHz (min.)
DC resistance (max.)	0.9 Ohms	
Insertion loss (max.)	1.0 dB	0 MHz to 65 MHz
HiPot (min.	1500 Vrms	

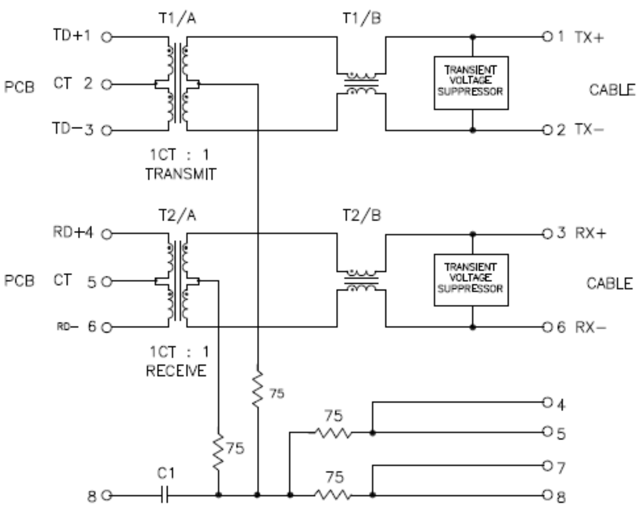
Ethernet design using RJ-45 with integrated magnetics

The Ethernet interface can be implemented with a single part by using an integrated part like TE Connectivity's 6605767-1 which has magnetics, common mode choke, termination and transient voltage suppression fully integrated in one part.

RJ-45 drawing



JX10-0006NL schematic



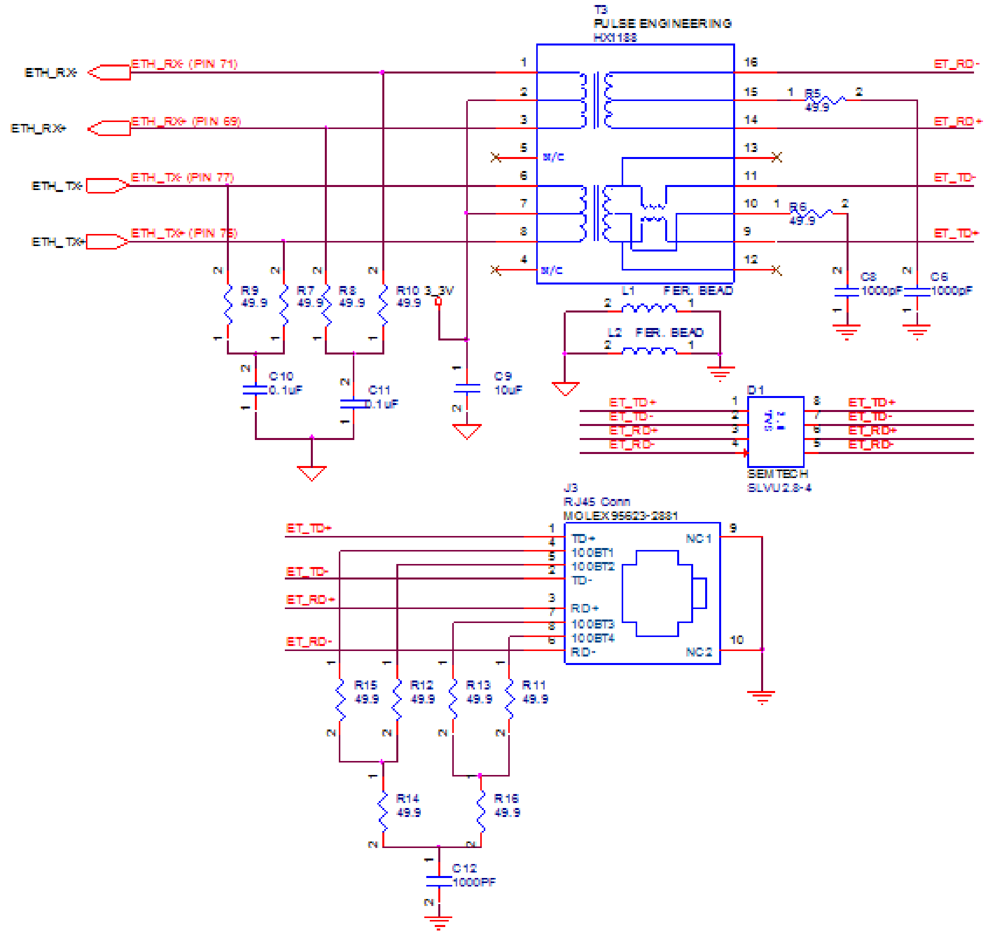
Electrical characteristics

Parameter	Specifications	
Insertion loss	100 kHz	1-125 MHz
	-1.2 dB max.	-0.2–0.002*f^1.4 db max.
Return loss (Z out = 100 Ohm +/- 15%)	0.1–30 MHz: 30–60 MHz: 60–80 MHz:	-16 dB min. $-10+20*LOG_{10}$ (f/60 MHz dB min.) -10 dB min.
Inductance (OCL) (Media side -40°C + 85°C)	350 uH min.	Measured at 100 kHz, 100 mVRMS and with 8 mA DC bias)
Crosstalk, adjacent channels	1 MHz	10-100 MHz
	-50 dB min.	$-50+17*LOG_{10}$(f/10) dB min.
Common mode rejection radio	2 MHz	30–200 MHz
	-50 dB min.	$-15+20*LOG_{10}$ (f/200) dB min.
DC resistance 1/2 winding	0.6 Ohms max.	
DC resistance imbalance	+/- 0.065 Ohms max. (center tap symmetry)	
input - output isolation	1500 Vrms min. at 60 seconds	

Ethernet design using discrete components

For maximum flexibility, a system integrator may choose to implement the Ethernet using discrete parts. The design below shows an example of such a design. It includes the Ethernet magnetics, termination of unused lines as well as surge protection. The magnetics used is a Pulse Engineering HX1188. Surge protection is provided by a Semtech SLVU2.8-4. In order to meet electrical isolation requirements, it is recommended to use capacitors with a greater than 2 kV breakdown voltage.

Ethernet schematic



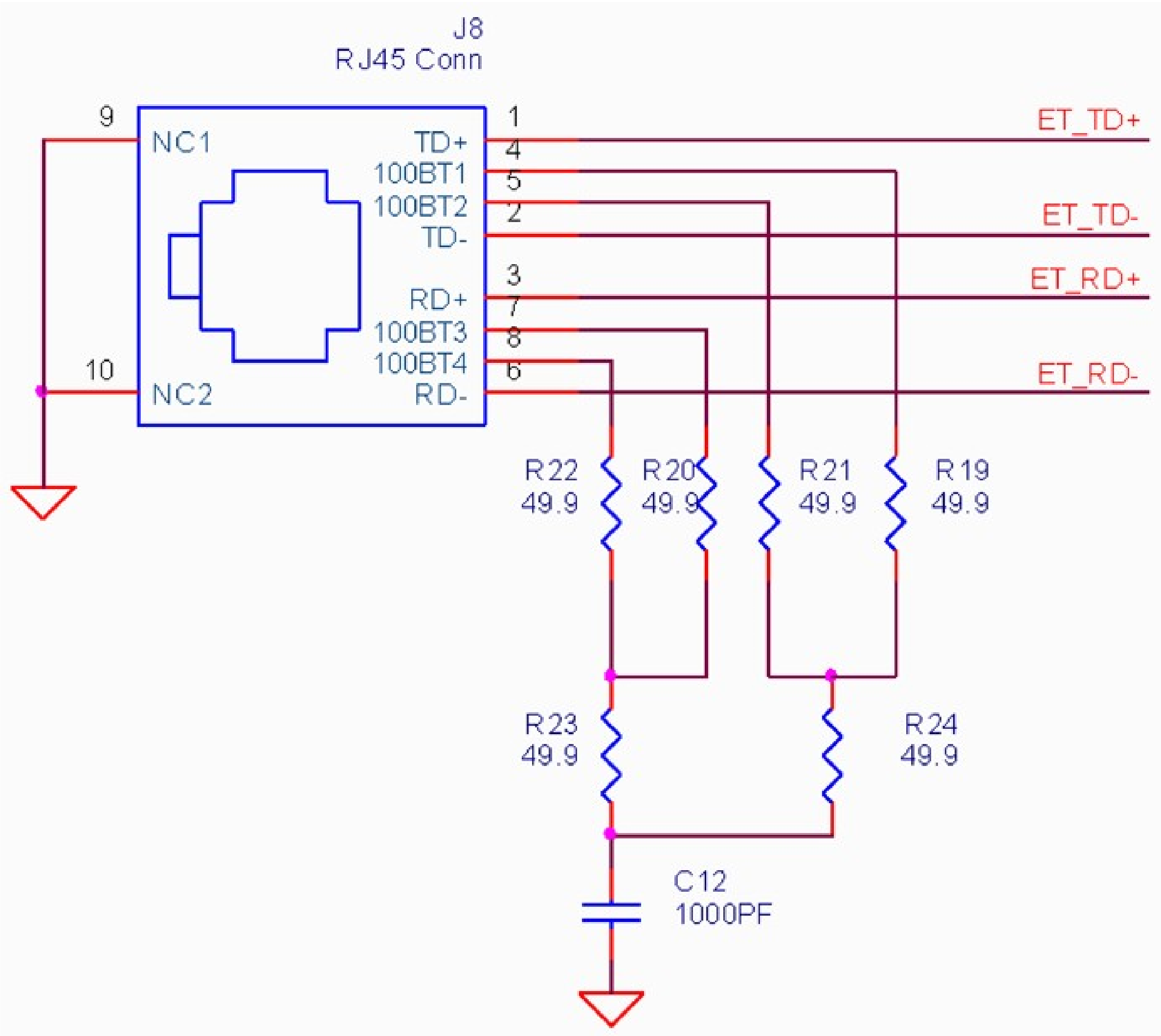
Part Reference	Value
C4–C6	1000pF 2 kV
C3	10 uF X5R 6.3 V
D1	SEMTECH SLVU2.8–4
J1	RJ45 Conn
L1, L2	Ferrite Bead
R1–R11	49.9 0402 1%
T1	Pulse engineering HX1188

Ethernet routing

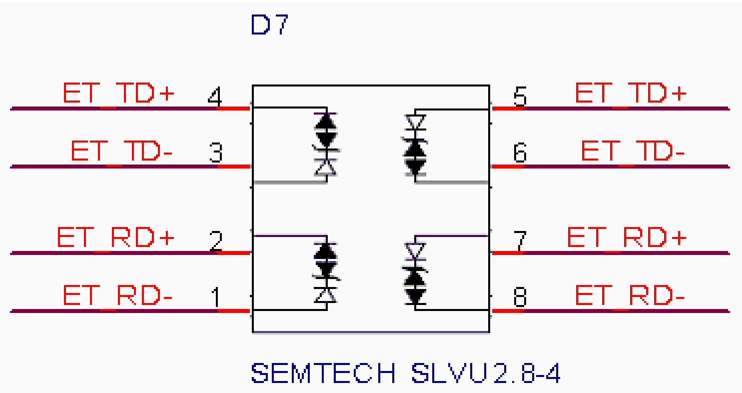
The distance from the BD940 connector, the Ethernet connector and the magnetics should be less than 2 inches. The distance from the RJ-45 and the magnetics should be minimized to prevent conducted emissions issues. In this design, the chassis ground and signal ground are separated to improve radiated emissions. The integrator may choose to combine the ground. The application note from the IC vendor is provided below for more detailed routing guidelines.

BD940-INS/BD99X Ethernet design considerations

The BD940-INS and BD9XX board series have their own magnetics, therefore, the Ethernet interface can be implemented using only a RJ-45 connector, and termination discretes. See design example below:



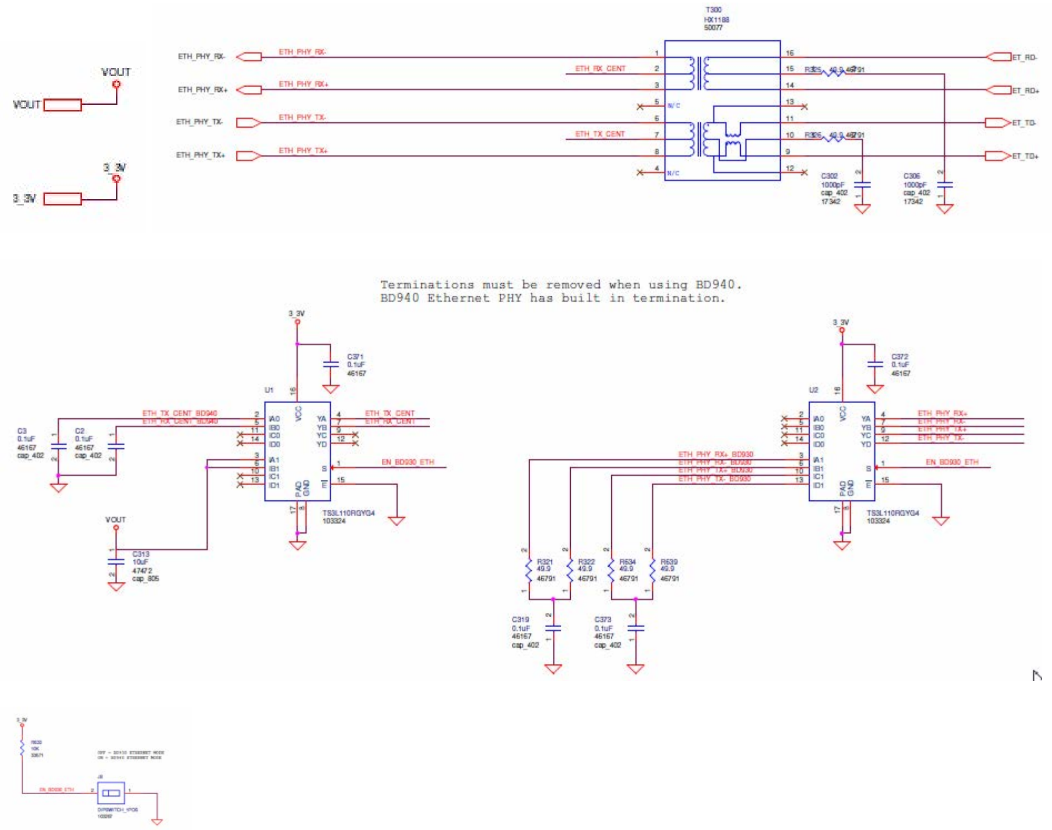
Optional surge protection is provided by a Semtech SLVU2.8-4. To meet electrical isolation requirements, Trimble recommends using capacitors with a greater than 2 kV breakdown voltage.

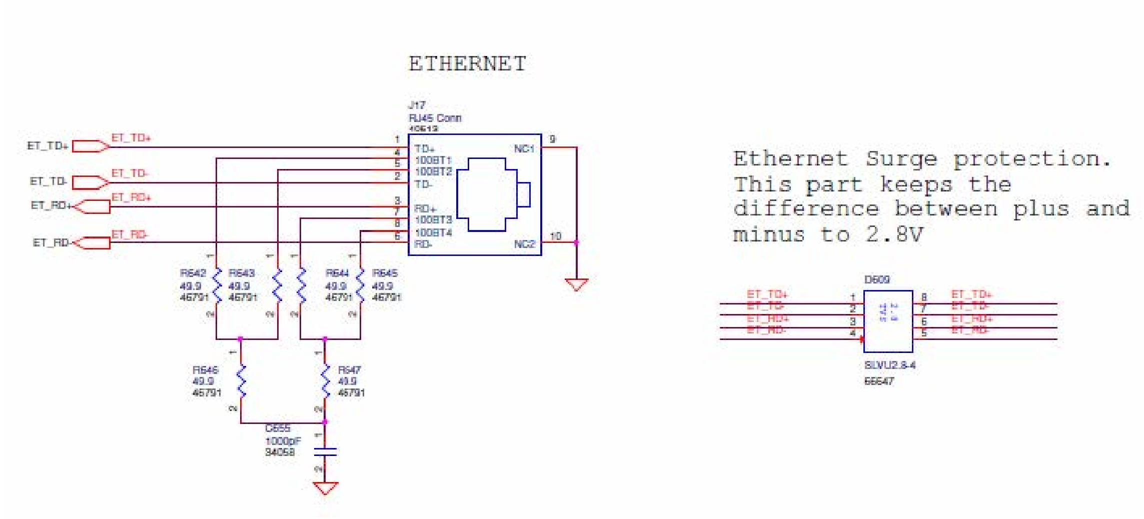


Evaluation board Ethernet schematics

BD940 evaluation board

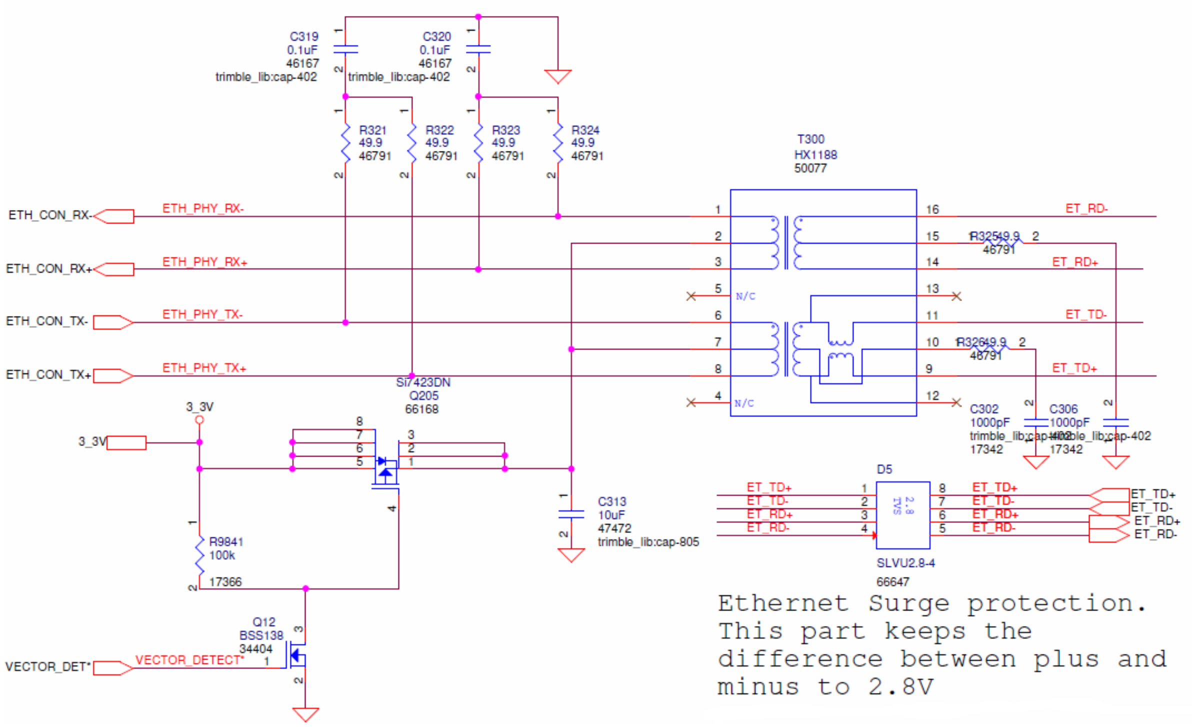
The evaluation board has the necessary magnetics to run the Ethernet interface. Below are the schematics of the Ethernet implementation on the BD940 evaluation board:

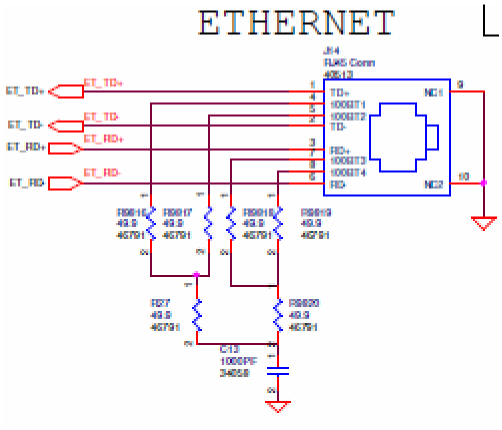




BD940-INS evaluation board

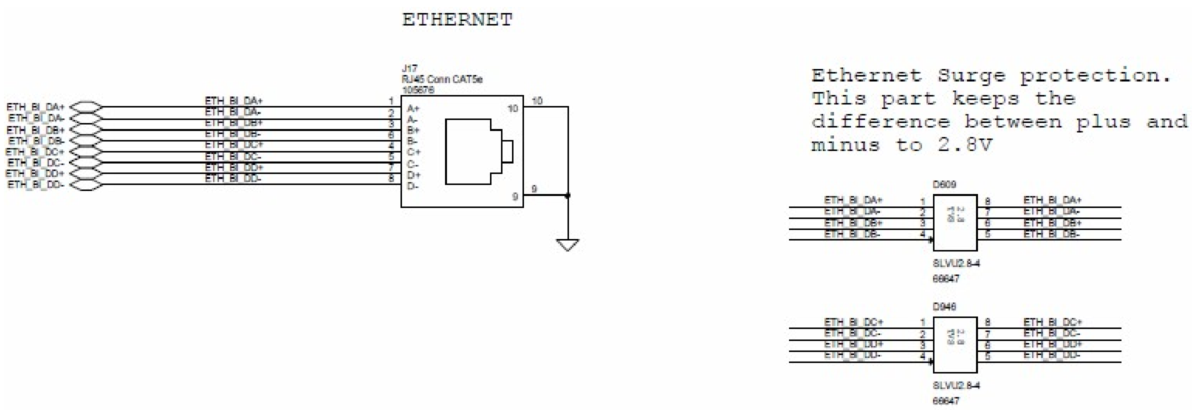
The BD940-INS has its own internal magnetics. The evaluation board also has Ethernet magnetics and in order to have both in series, the choke is left flowing.



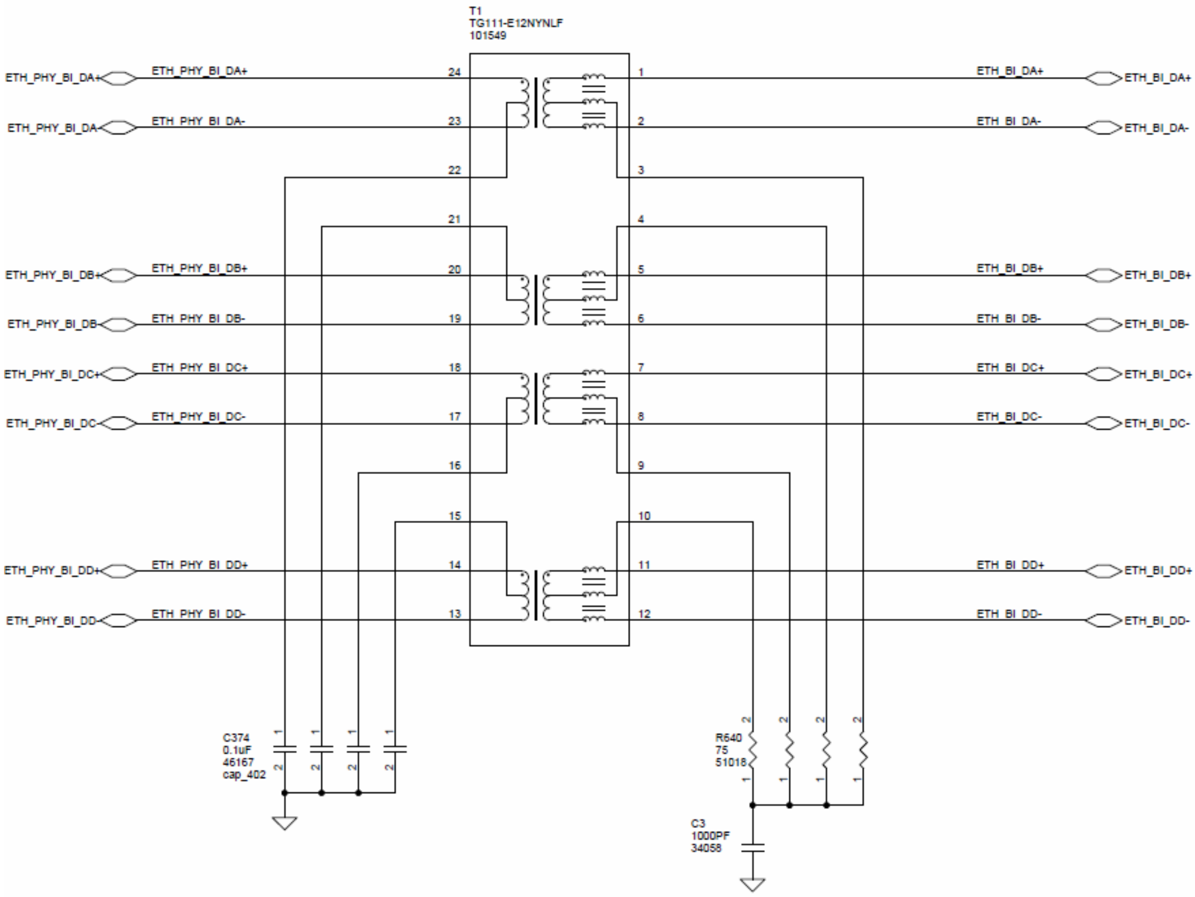


BD99X evaluation board

The BD99X series of boards has its own internal magnetics. The following details the implementation of the evaluation board Ethernet circuitry;



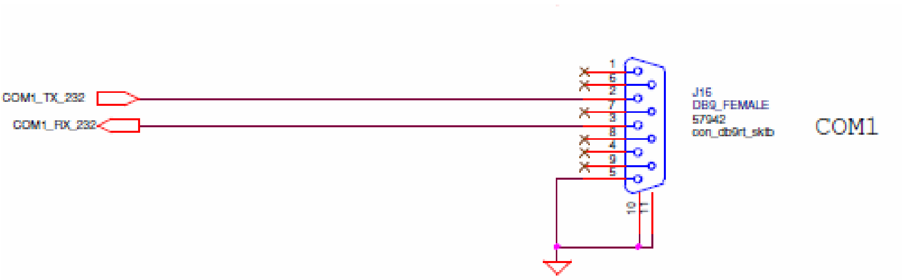
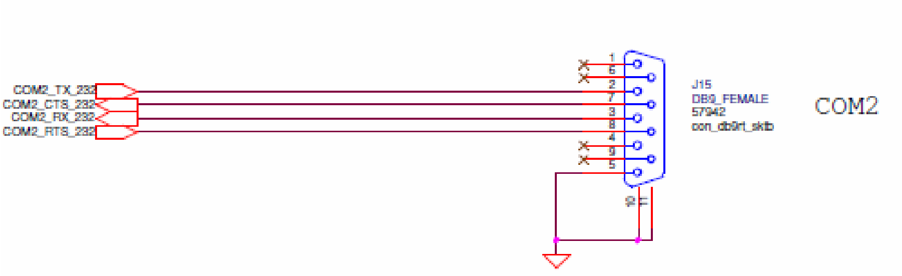
Ethernet magnetics



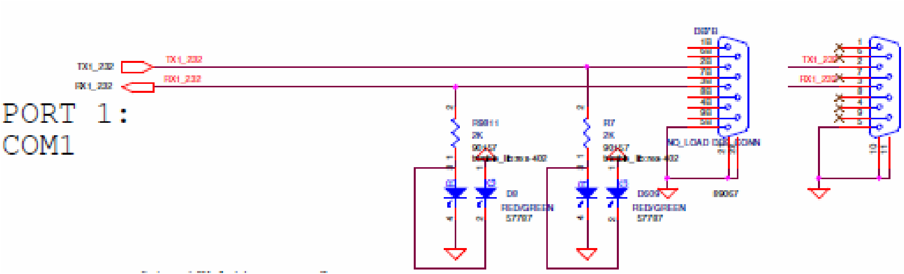
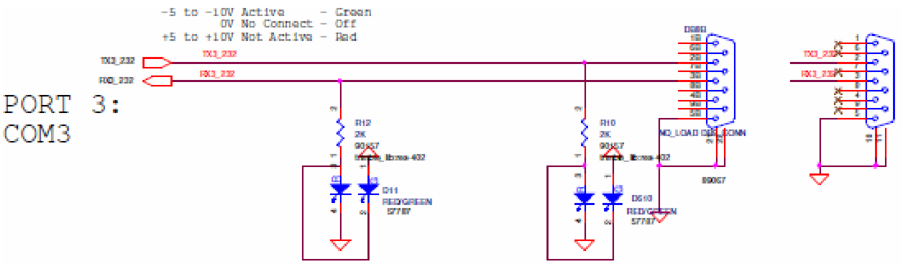
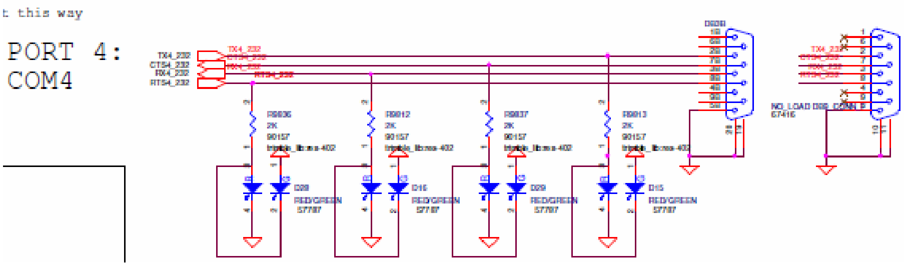
3

Serial Port

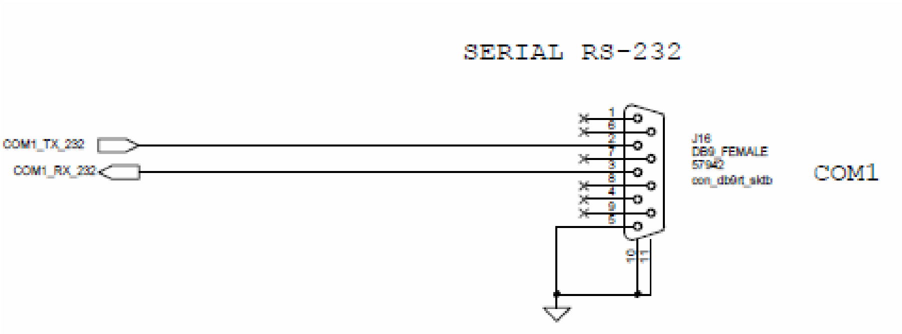
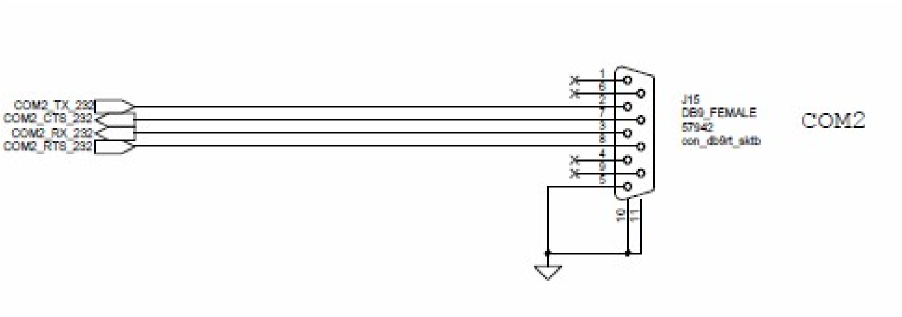
BD940 evaluation board

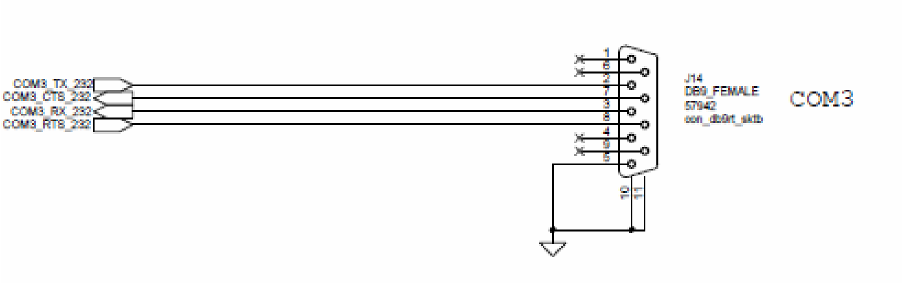
Item	Description
Port 1 (no flow control)	COM 1 is already at RS-232 level and already has 8 kV contact discharge/15 kV air gap discharge ESD Protection. This is labeled Port 1 on the I/O board. 
Port 2 (with flow control)	COM 2 is at 0-3.3 V TTL. This port has RTS/CTS to support hardware flow control. If the integrator needs this port to be at RS-232 level, a proper transceiver powered by the same 3.3 V that powers the receiver needs to be added. For development using the I/O board, this COM port is already connected to an RS-232 transceiver. This is labeled Port 2 on the I/O board. 

BD940-INS evaluation board

Item	Description
Port 1 (no flow control)	COM 1 is already at RS-232 level and already has 8 kV contact discharge/15 kV air gap discharge ESD Protection. This is labeled Port 1 on the I/O board. 
Port 3 (no flow control)	COM 3 is already at RS-232 level and already has 8 kV contact discharge/15 kV air gap discharge ESD Protection. This is labeled Port 3 on the I/O board. 
Port 4 (with flow control)	COM 4 is at 0-3.3 V TTL. This port has RTS/CTS to support hardware flow control. If the integrator needs this port to be at RS-232 level, a proper transceiver powered by the same 3.3 V that powers the receiver needs to be added. For development using the I/O board, this COM port is already connected to an RS-232 transceiver. This is labeled Port 4 on the I/O board. 

BD99x evaluation board

Item	Description
Port 1 (no flow control)	COM 1 is already at RS-232 level and already has 8 kV contact discharge/15 kV air gap discharge ESD Protection. This is labeled Port 1 on the I/O board. 
Port 2 (with flow control)	COM 2 is at 0-3.3 V TTL. This port has RTS/CTS to support hardware flow control. If the integrator needs this port to be at RS-232 level, a proper transceiver powered by the same 3.3 V that powers the receiver needs to be added. For development using the I/O board, this COM port is already connected to an RS-232 transceiver. This is labeled Port 2 on the I/O board. 
Port 3 (with flow control)	COM 3 is at 0-3.3 V TTL. This port has RTS/CTS to support hardware flow control. If the integrator needs this port to be at RS-232 level, a proper transceiver powered by the same 3.3 V that powers the receiver needs to be added. For development using the I/O board, this COM port is already connected to an RS-232 transceiver. This is labeled Port 3 on the I/O board.

Item	Description
	

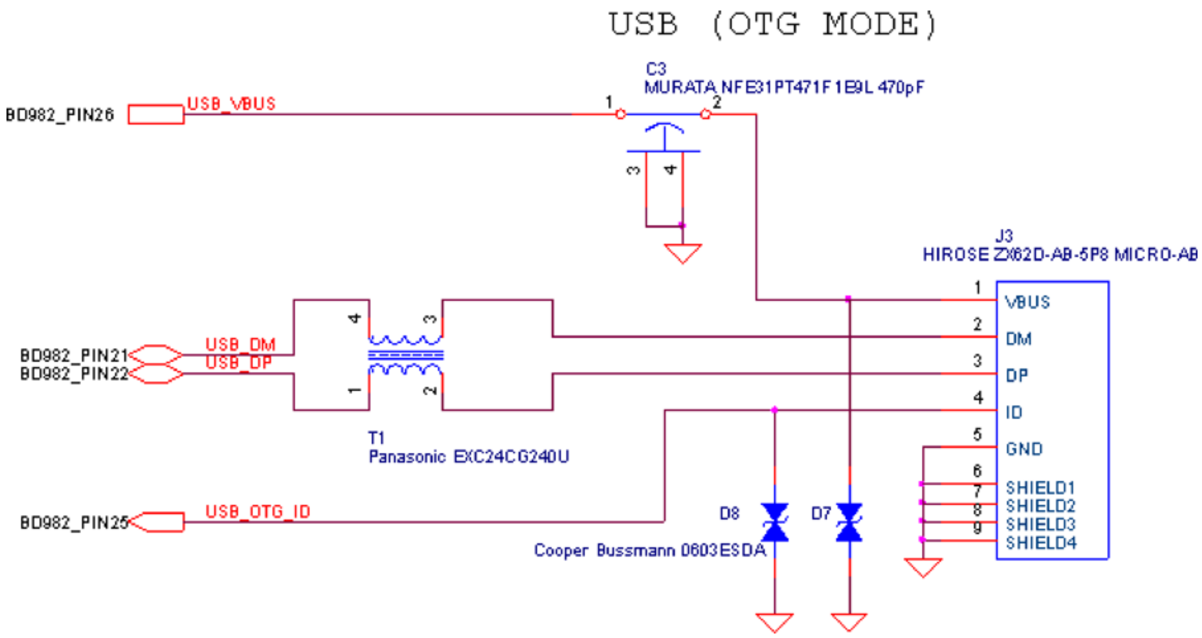
4

USB

The CPU of the receiver has an integrated PHY that supports both USB 2.0 Device and Host configuration at low speed, full speed, and high speed. In Host mode, the receiver supplies 5 V to a USB device, such as a memory stick. In Device mode, the receiver behaves like an external storage device to a computer.

USB OTG reference design

To be OTG-compliant, the connector must be MICRO AB. An OTG-compliant cable has A and B ends. When the B-side of the cable is inserted, the ID pin is not connected (floating) and the receiver enters Device mode through a pull-up resistor. The A-side of the cable connects the ID pin to ground, which enables Host mode on the receiver.



To reduce EMI, place a USB 2.0 compliant common mode choke on the data lines. To ensure best EMI performance, locate the choke near the USB MICRO AB connector. Trimble recommends that you use an L-C-L type EMI filter for the output power.

For product robustness and protection, place ESD protection diodes on both the USB_VBUS and USB_OTG_ID lines. The receiver has internal high-speed ESD protection on the USB data lines.

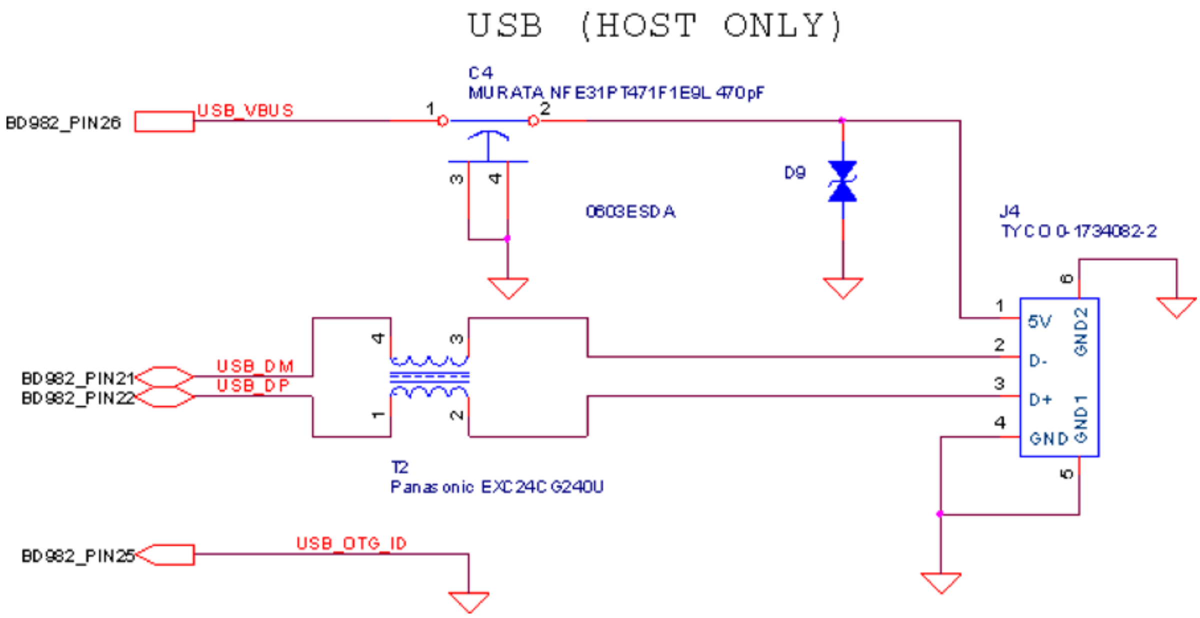
To ensure best USB high-speed performance, carefully consider PCB routing and placement practices:

- Place components so the trace length is minimized.
- Do not have stubs on data lines more than 0.200".
- Route data lines differentially but as parallel as possible.
- Data lines must be controlled to 90 Ohms differential impedance, and 45 Ohms single-ended impedance.
- Route over continuous reference plane (either ground or power).

For more detailed information, refer to the *Intel High Speed USB Platform Design Guidelines*.

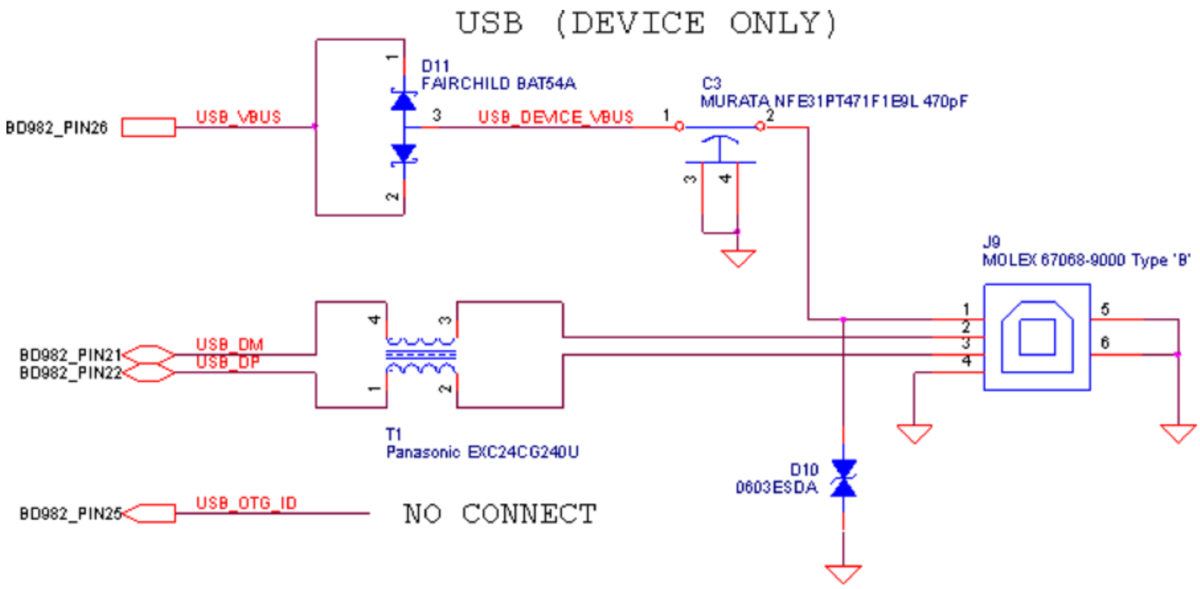
USB host-only reference design

For USB host-only support, a type-A connector is required. Since the receiver dos not support dynamic role switching, the ID pin should be grounded on the receiver. In Host mode, the receiver supplies nominal 5 V output at 500 mA to the USB device.



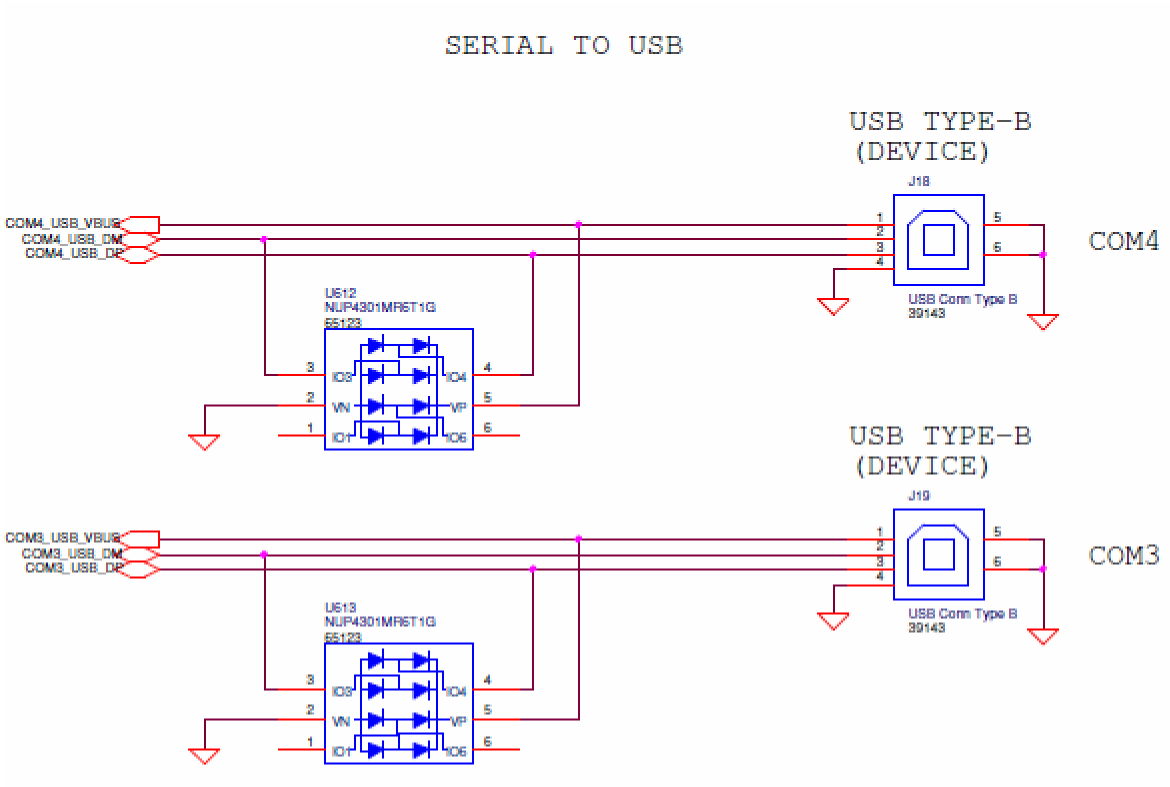
USB device-only reference design

For device-only operation, the USB_OTG_ID pin is left floating. For reference, the receiver has an internal 10 K Ohm pull-up to 3.3 V. In this mode, the USB_DEVICE_VBUS is used only by receiver to detect if host power is connected.

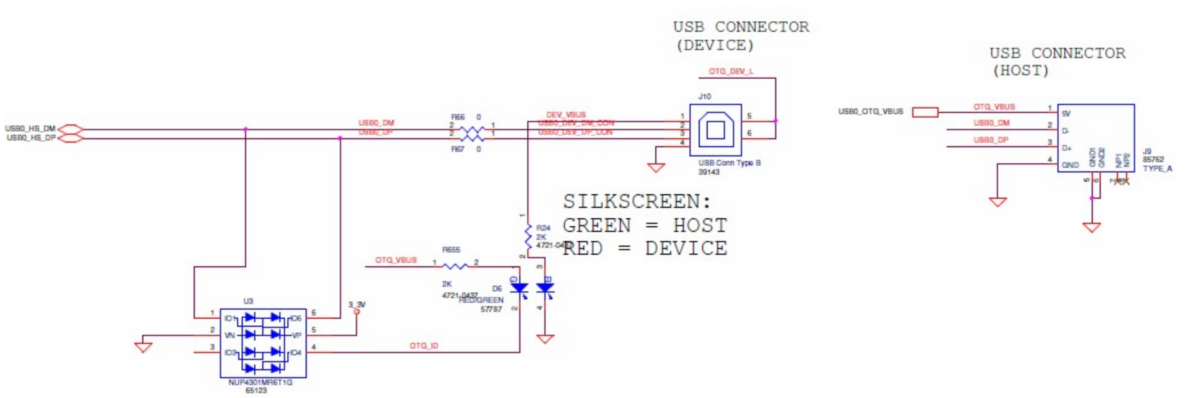


BD940 evaluation board

The BD940 evaluation board has two Serial – USB ports. The ports are named COM3 and COM4. The USB connectors are Type-B connectors. Below is the schematic for the two Serial – USB communication ports.

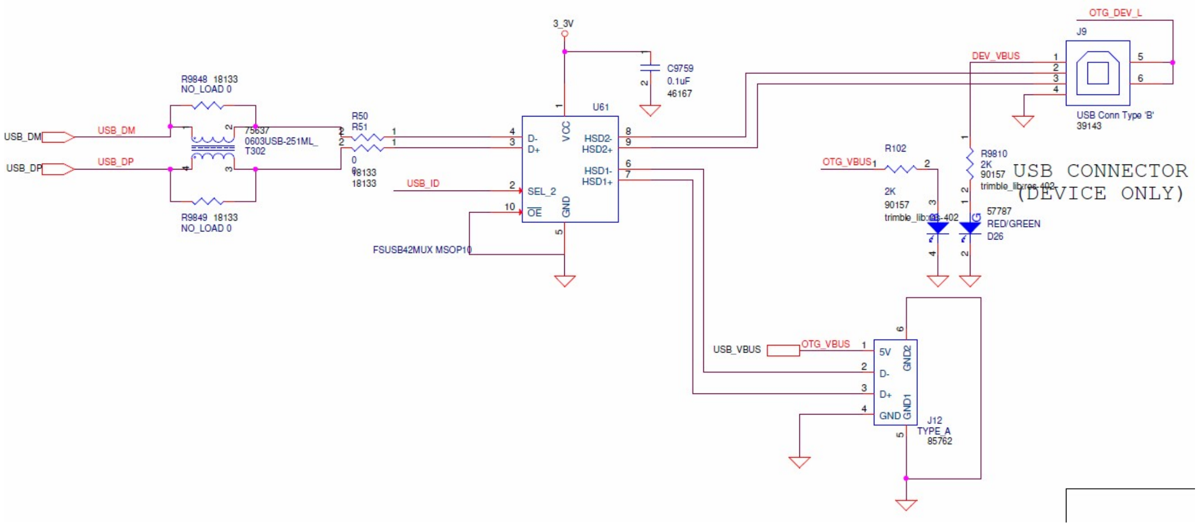


The BD940 evaluation board is also equipped to handle both USB Device and Host. There are two USB ports available on the evaluation board that provide this functionality. Below is the schematic that details the connections involved with both USB modes.



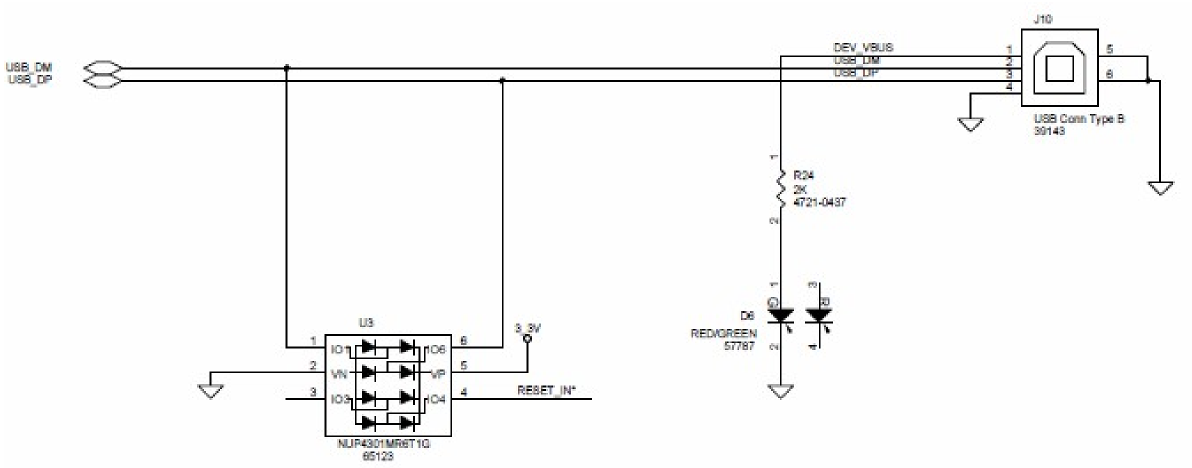
BD940-INS evaluation board

The following is the schematic for the USB Type-B (device only) connector on the BD940-INS evaluation board:



BD99x evaluation board

The BD99X evaluation board has one USB device port. The following is the schematic for this communication port:



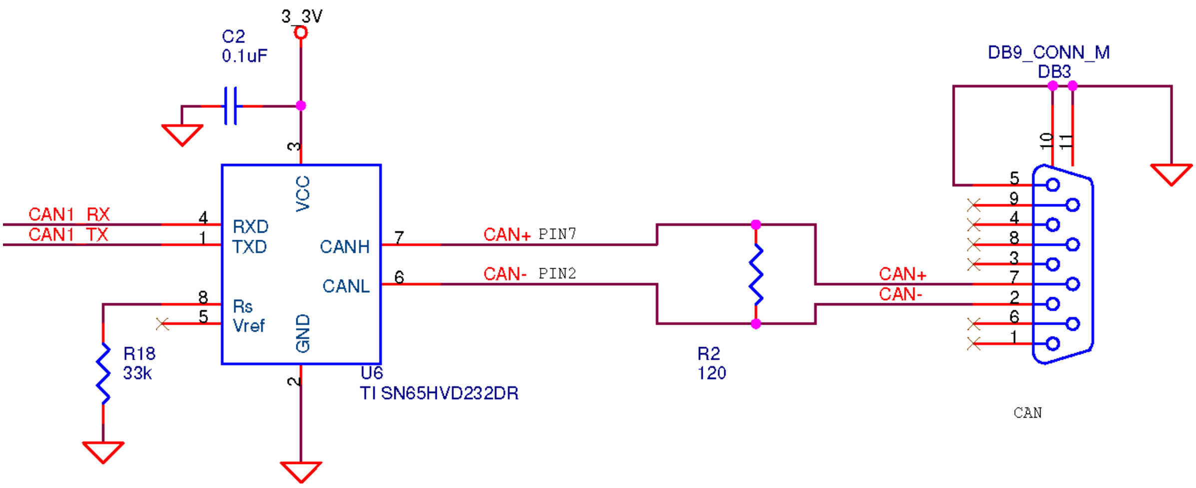
5

CAN

BD940

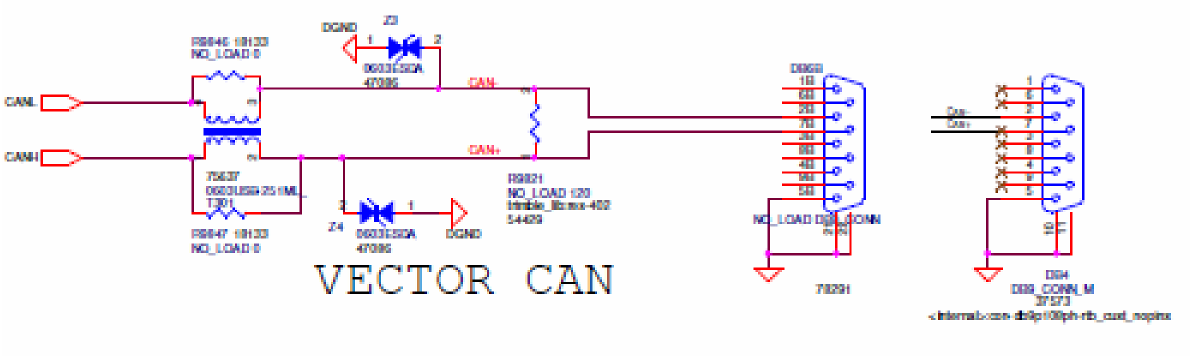
The BD940 module does not have the capability for CAN. To output CAN messages, the BD940 needs an external CAN transceiver.

The following figure shows a typical implementation with a 3.3 V CAN transceiver. It also shows a common mode choke as well as ESD protection. A 5 V CAN Transceiver can be used if proper level translation is added.



BD940-INS evaluation board CAN port

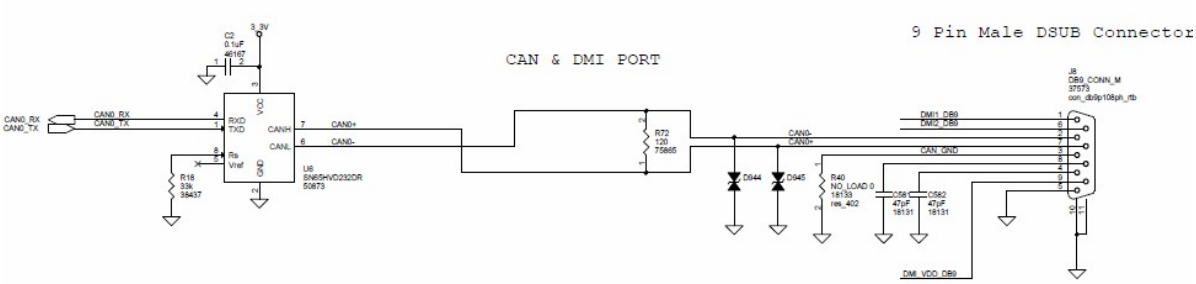
For CAN development using the BD940-INS evaluation board, the provided CAN port is already connected to a CAN transceiver. This port is marked as “CAN” on the board. The connector used is a standard 9-pin DB9 connector.

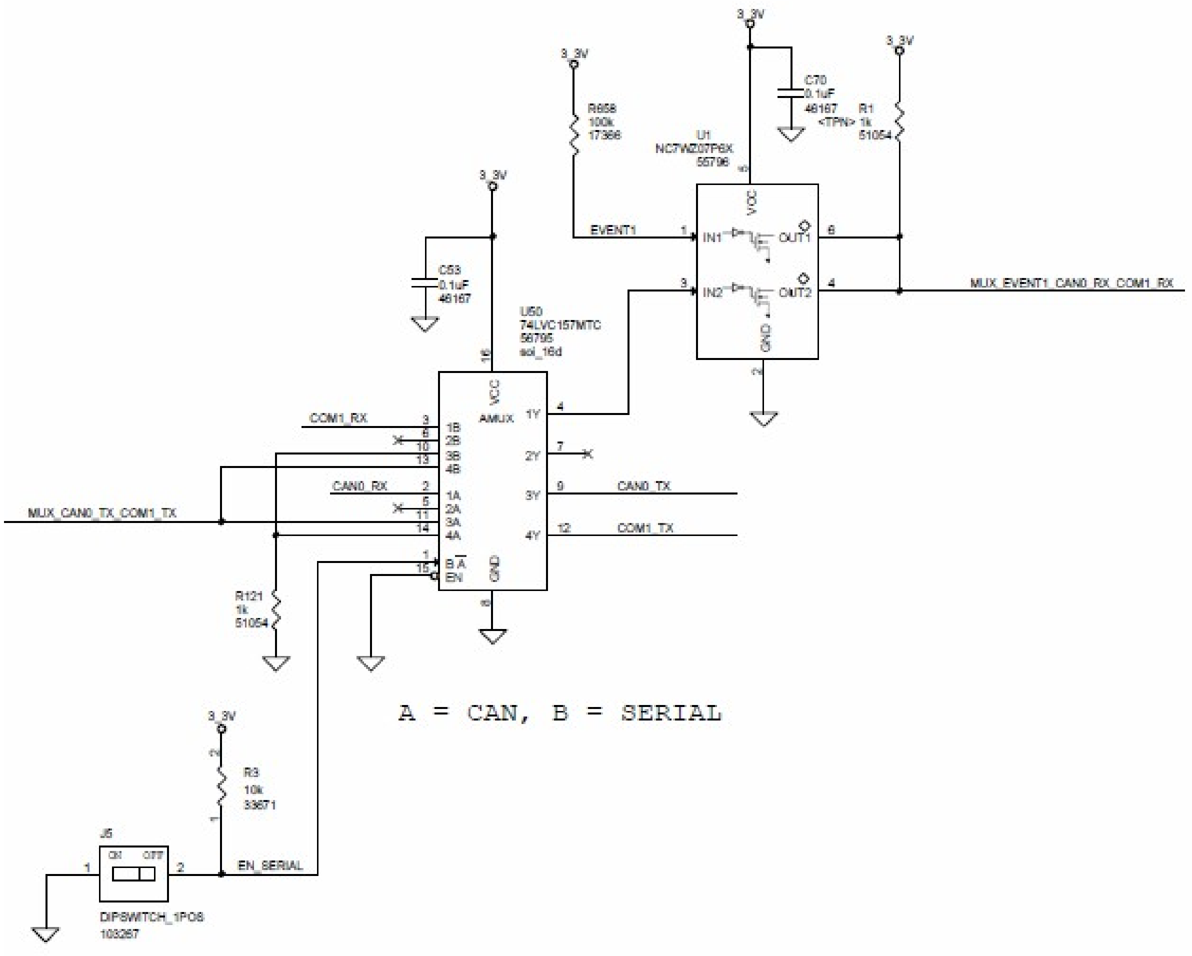


BD99X evaluation board CAN port

Com 4 is at 0-3.3 V TTL and is multiplexed with CAN. The receive line is also multiplexed with Event 1. The integrator must have a receiver configured to use the CAN port in order to use this port as a serial port. The functionality cannot be multiplexed in real time.

For development using the I/O board, this com port is already connected to a CAN transceiver. This is labeled CAN on the I/O board. J5, labeled ‘CAN’ and ‘SERIAL’, must be set to CAN. There shouldn't be anything connected to TP6, labeled Event 1.




3_3V
R658
100k
17366
U1
NC7WZ07P6X
55796
C70
0.1uF
46167
R1
<TPN> 1k
51054
EVENT1
IN1
OUT1
IN2
OUT2
VCC
GND
MUX_EVENT1_CAN0_RX_COM1_RX
C53
0.1uF
46167
U50
74LVC157MTC
56795
soi_16d
AMUX
COM1_RX
1B
2B
3B
4B
CAN0_RX
1A
2A
3A
4A
B A
EN
1Y
2Y
3Y
4Y
CAN0_TX
COM1_TX
MUX_CAN0_TX_COM1_TX
R121
1k
51054
A = CAN, B = SERIAL
R3
10k
33671
J5
ON
OFF
EN_SERIAL
DIPSWITCH_1POS
103267

6

LED Control Lines

Item	Description
Driving LEDs	The outputs are 3.3 V TTL level with a maximum source/sink current of 4 mA. An external series resistor must be used to limit the current. The value of the series resistor in Ohms is determined by: (3.3-Vf)/(If) > Rs > (3.3 V - Vf)/(.004) Rs = Series resistor If = LED forward current, max typical If of the LED should be less than 3mA Vf = LED forward voltage, max typical Vf of the LED should be less than 2.7V Most LEDs can be driven directly as shown in the circuit below: LEDs that do not meet If and Vf specification must be driven with a buffer to ensure proper voltage level and source/sink current.
Power LED	This active-high line indicates that the unit is powered on.
Satellite LED	This active-high line indicates that the unit has acquired satellites. A rapid flash indicates that the unit has less than 5 satellites acquired while a slow flash indicates greater than 5 satellites acquired. This line will stay on if the unit is in monitor mode.
RTK Correction	A slow flash indicates that the unit is receiving corrections. This will also flash when the unit is in monitor mode.

7

Event Input for the BD9xx Using the Evaluation Board

This topic describes how to condition and analyze event input signals when using the BD9xx evaluation boards. This knowledge also applies to the customers’ implementation of event inputs on their carrier board for the BD9xx.

Useful links:

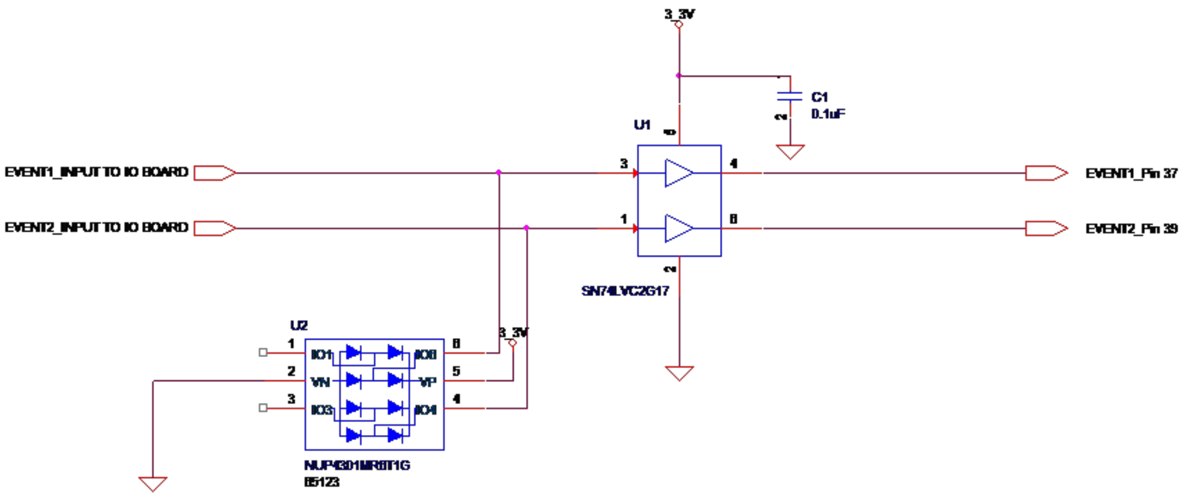
- For information about the web interface, refer to the Web Interface section of the BD9xx User Guide.
- For Event 1 and Event 2 information, see page 144 in Revision E of the User Guide.

Event

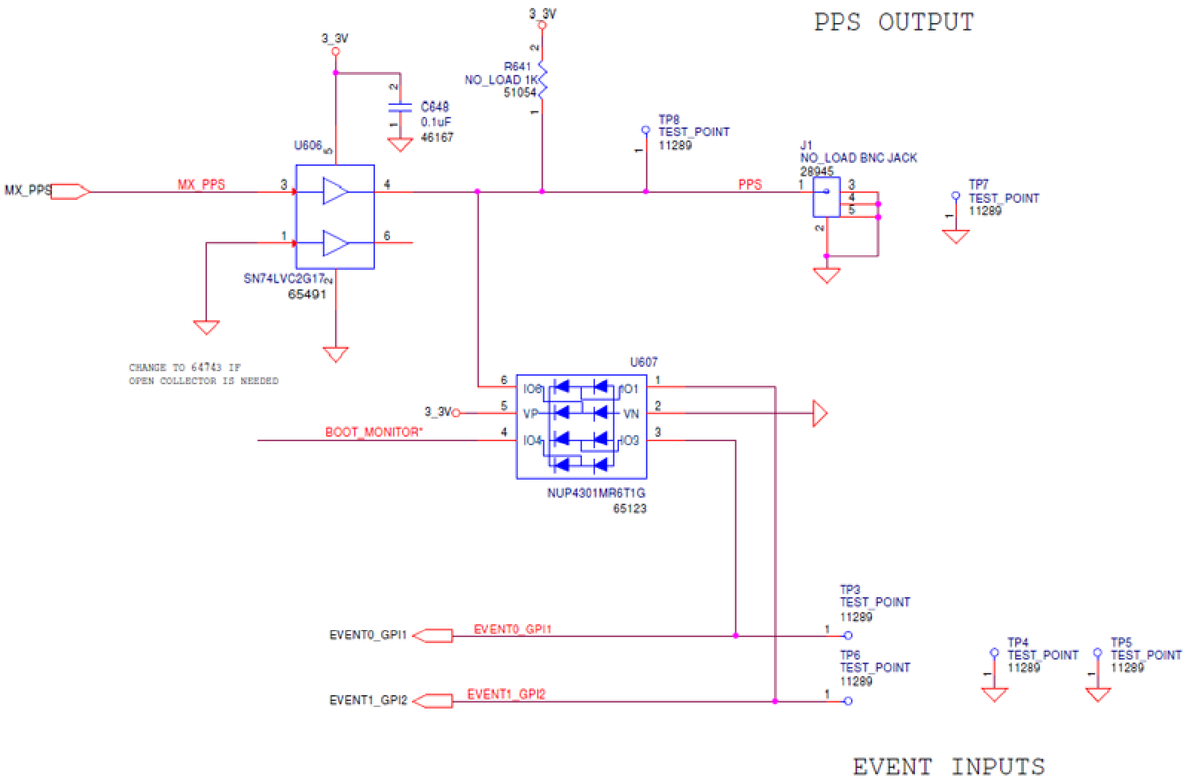
Item	Description
Event 1	Pin 8 is dedicated as an Event_In pin. This is a TTL only input; it is not buffered or protected for any inputs outside of 0 V to 3.3 V. It does have ESD protection. If the system requires event to handle a voltage outside this range, the system integrator must condition the signal prior to connecting to the unit.
Event 2	Event 2 is multiplexed with COM3_RX and CAN_RX. The default setting is to have this line set to COM3_RX. The Event 2 must be enabled in order to use Event2. When using the 63494 Development interface board, the user must not connect anything to Port 3 and the CAN port when using Event 2. The Com3 level selection switch is ignored when Event 2 is selected. This is a TTL only input; it is not buffered or protected for any inputs outside of 0 V to 3.3 V. It does have ESD protection. If the system requires event to handle a voltage outside this range, the system integrator must condition the signal prior to connecting to the unit.

Event schematics of the BD9xx evaluation PCB

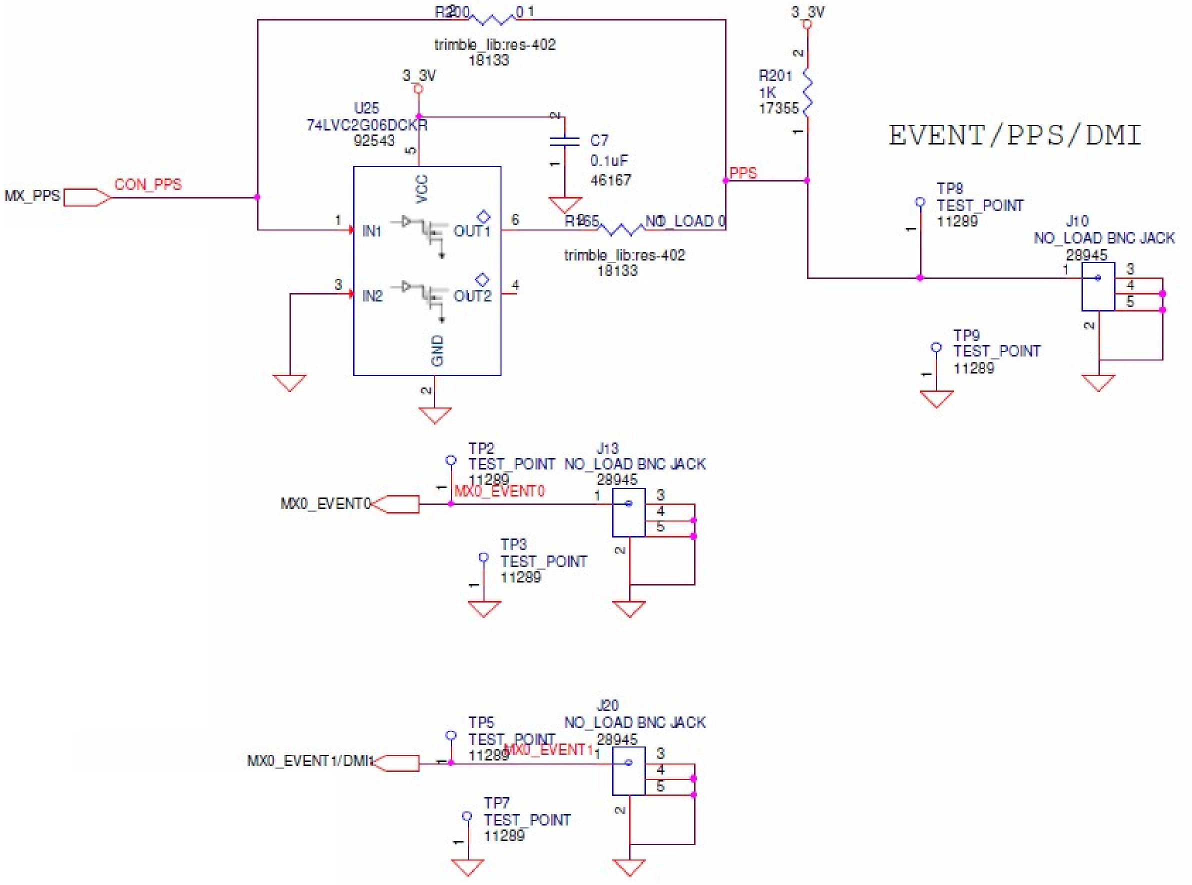
Trimble recommends adding a Schmitt trigger and ESD protection to the Event_In pin. This prevents any "ringing" on the input from causing multiple and incorrect events to be recognized.



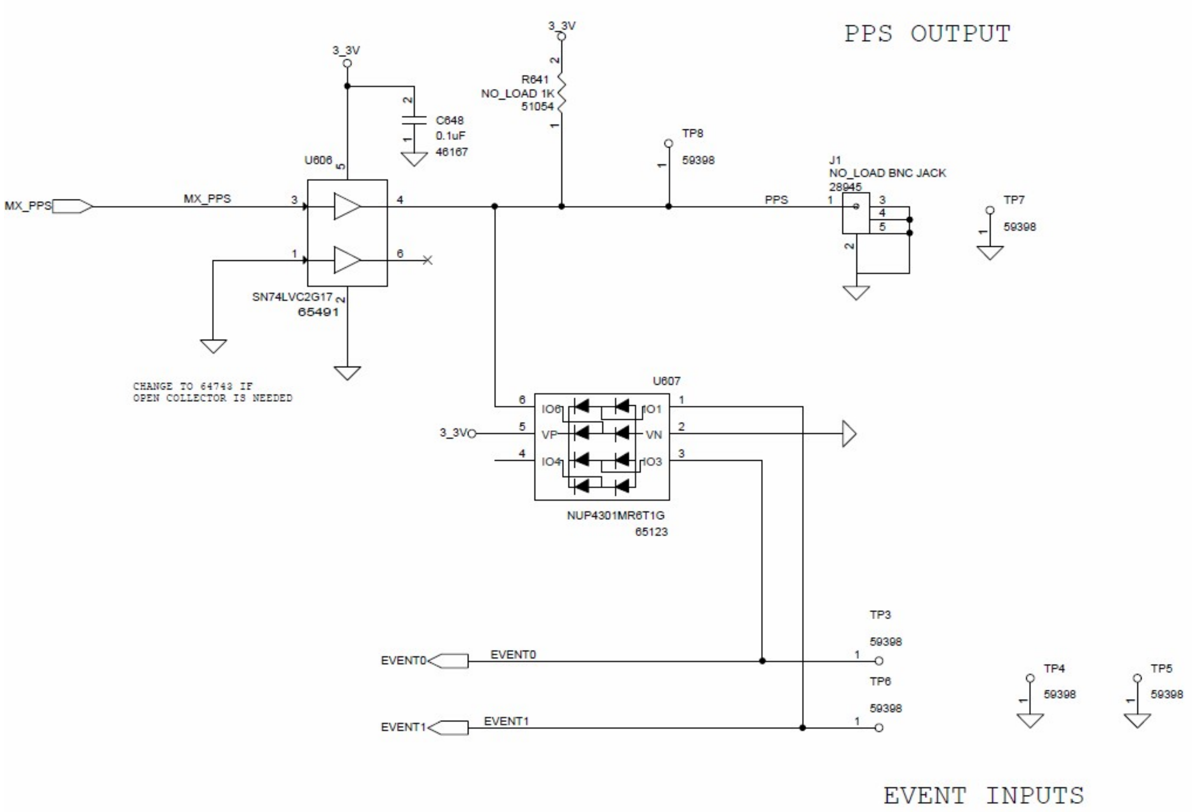
PPS output and event inputs BD940



PPS output and event inputs BD40-INS



PPS output and event inputs BD990/BD992/BD992-INS



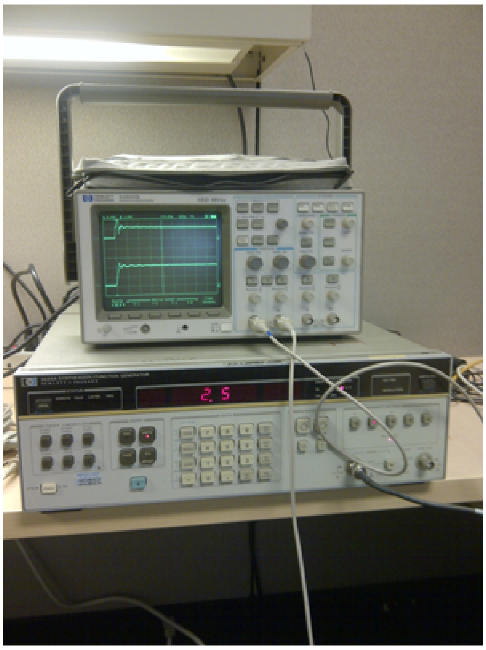
Event (0) 1PPS Input example

This section illustrates an example hardware setup, inputting a 1 PPS signal from an external source.

The specifications of the input signal are:

- Frequency: 1 Hz
- Amplitude: 2.5 V DC P-P (Peak to Peak) (within the specified 3.3 maximum voltage)
- DC offset: 1.5 V DC

Hardware to generate and measure the input signal



Issues of conditioning the input voltage signal

The following examples illustrate the principle of conditioning the input voltage signal:

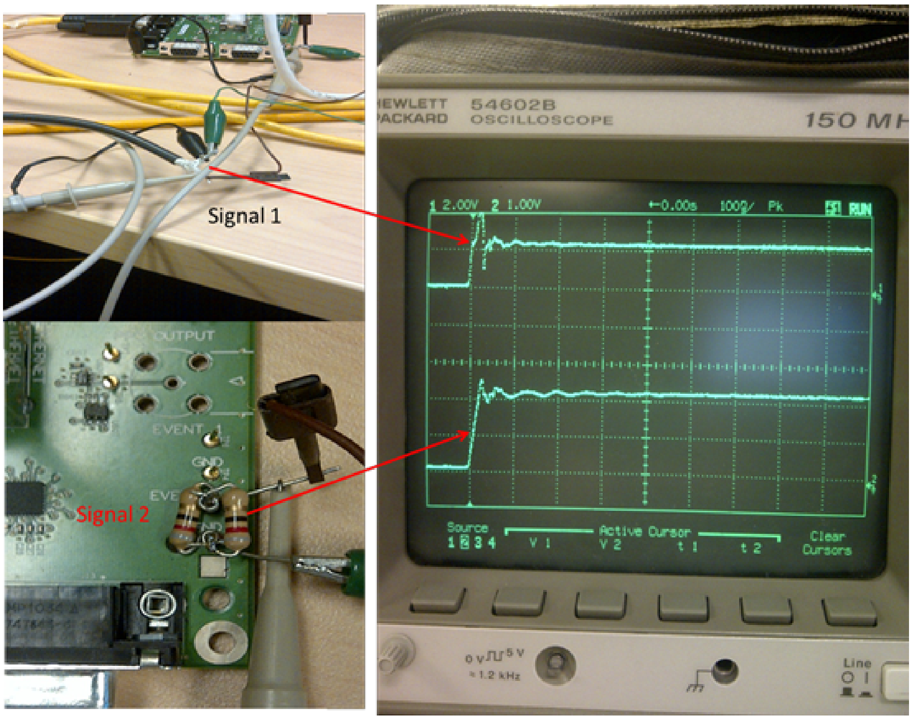
- In this example, the oscilloscope is measuring the signal in two places. When probing the signal (Signal 1), it measures an output voltage overshoot over 4 V DC, then settles to about 2.5 V DC. The "overshoot" is above the maximum allowable input voltage of 3.3 V DC. To condition the input voltage on the I/O board, add two 82 Ohm resisters in parallel, which gives about 41 Ohms of resistance to correct for the overshoot. The input voltage to the Event (0) 1 after the 41 Ohms of resistance is measured at Signal 2.
- In this example, the measured signal at Signal 2 may have a 'ringing' characteristic after the rising edge. This 'ringing' may trigger 2 Event Inputs into the system, causing 2

Event Inputs to be triggered in the firmware and logged. This signal characteristic is undesirable. See Verifying the Event In Data using the RT17/RT27 Protocol.

Oscilloscope Signal 1 and Signal 2

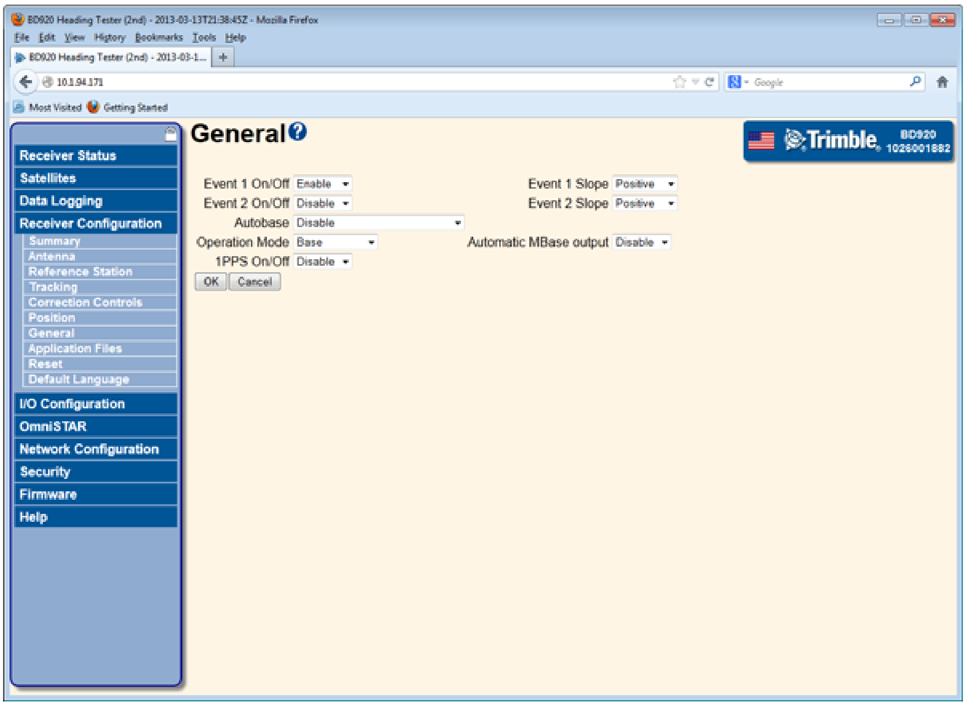
Signal 1: Vertical scale is 2-volt increments

Signal 2: Vertical scale is 1-volt increments



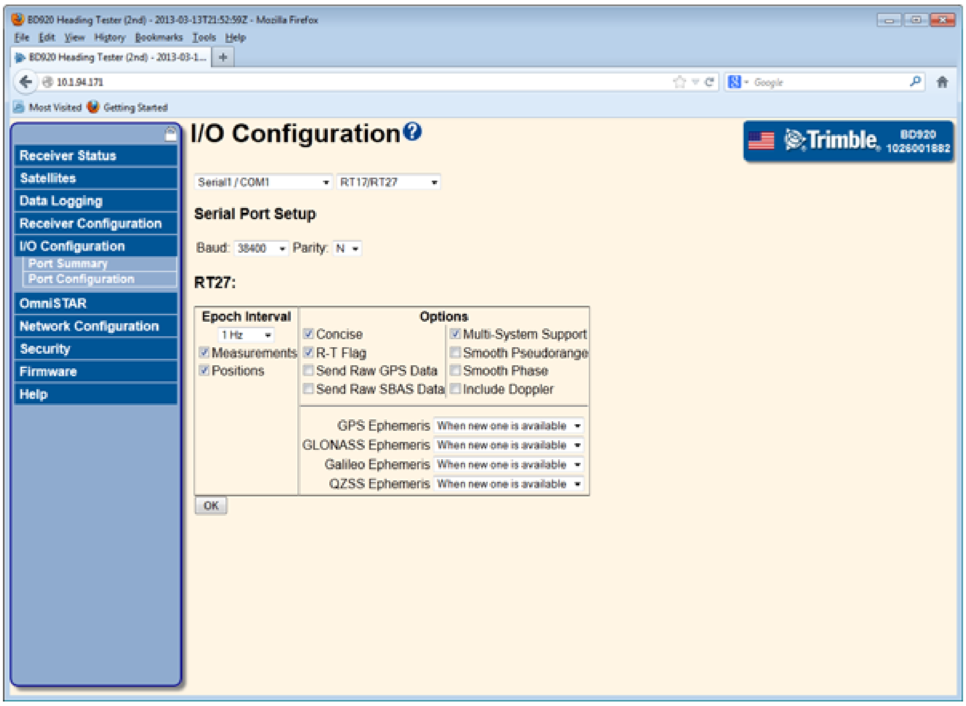
Enabling Event (0) 1 in the receiver firmware using the web interface

In the **General** page of the **Receiver Configuration** menu, enable the Event inputs:



Logging Event In using the RT17/RT27 protocol

In the **I/O Configuration** page, select how you want to collect the data. The example below shows COM1 with the defaults of 38400, Parity None (N). The RT17/27 Log is also enabled:



Verifying the Event In data using the RT17/RT27 protocol

Data events 1-second interval:

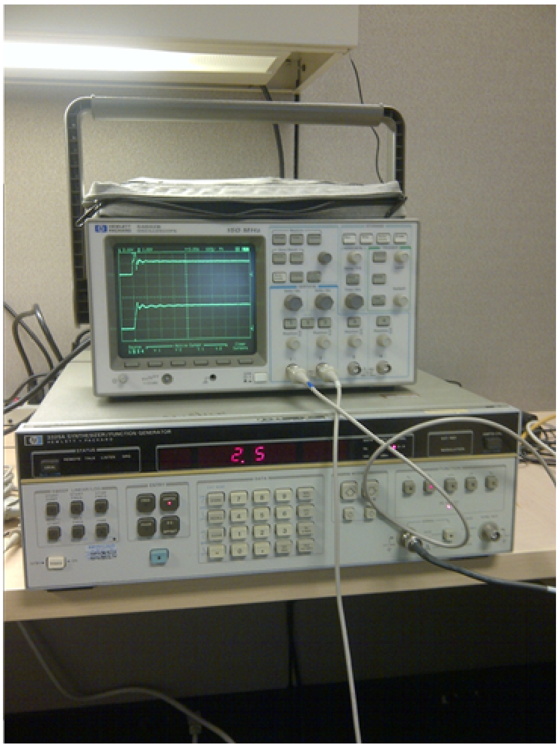
BD920 HW Event (0) 1 PPS
Input using a
HP Sig Gen 3325A
0 to 2.5 V Peak to Peak

Record Type: 2 (2: Event Mark)Page Number 1 of 1Reply Number: 37Record Interpretation Flags: 00000000 bit 0 set: Type 17/27 Concise format (if 0 then Enhanced format) bit 1 set: Enhanced Record with real-time flags and IODE

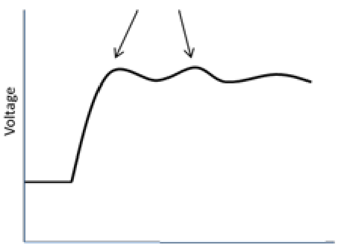
informationDecoding message: EMPTY MESSAGE STRING LENGTH IN VARIABLE: Event Source Event Source: 7 Event Port: 1 (1: 1st Event Port or Serial Port 1) Event Number: ***500*** GPS time: 2386***17***720.092215

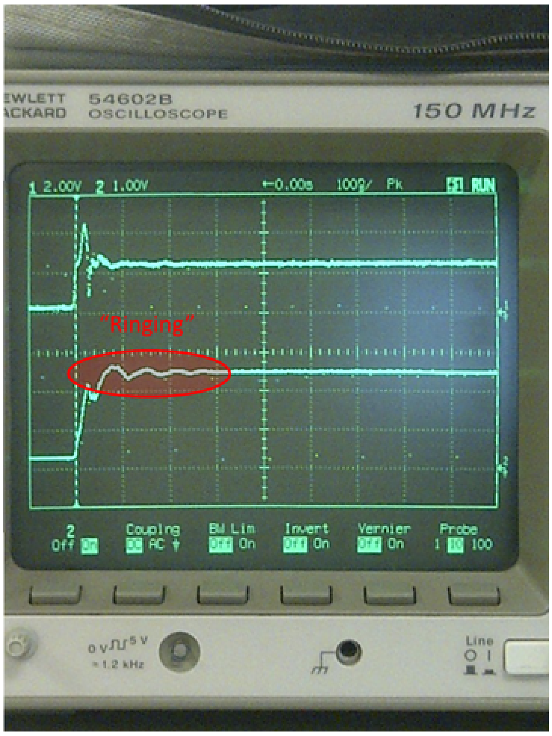
Record Type: 2 (2: Event Mark)Page Number 1 of 1Reply Number: 40Record Interpretation Flags: 00000000 bit 0 set: Type 17/27 Concise format (if 0 then Enhanced format) bit 1 set: Enhanced Record with real-time flags and IODE

informationDecoding message: EMPTY MESSAGE STRING LENGTH IN VARIABLE: Event Source Event Source: 7 Event Port: 1 (1: 1st Event Port or Serial Port 1) Event Number: ***501*** GPS time: 2386***18***720.171727



2 similar Voltage Peaks which could cause 2 Event Inputs to be triggered in the Firmware.





In the case where two events are recorded within 1-second interval:

- Record Type: 2 (2: Event Mark)
- Page Number 1 of 1
- Reply Number: 18
- Record Interpretation Flags: 00000000
- Decoding message:
- Event Source: 7
- Event Port: 1 (1: 1st Event Port or Serial Port 1)
- Event Number: 4
- GPS time: 442516134.467520 (ms)

In the case where there is too much ringing:

- Record Type: 2 (2: Event Mark)
- Page Number 1 of 1
- Reply Number: 19
- Record Interpretation Flags: 00000000
- Decoding message:
- Event Source: 7
- Event Port: 1 (1: 1st Event Port or Serial Port 1)
- Event Number: 5
- GPS time: 442516136.342442 (ms)
- Delta GPSTime between Events: 1.874921978 (ms)

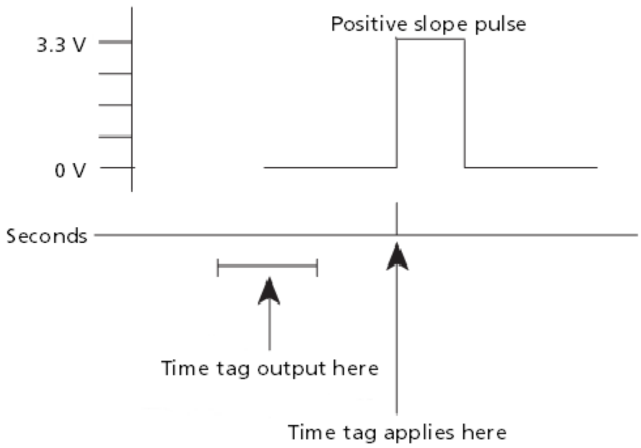
8

1PPS and ASCII Time Tag

The receiver can output a 1 pulse-per-second (1PPS) time strobe and an associated time tag message. The time tags are output on a user-selected port.

The leading edge of the pulse coincides with the beginning of each UTC second. The pulse is driven between nominal levels of 0.0 V and 3.3 V (see below). The leading edge is positive (rising from 0 V to 3.3 V). The receiver PPS out is a 3.3 V TTL level with a maximum source/sink current of 4 mA. If the system requires a voltage level or current source/sink level beyond these levels, you must have an external buffer. This line has ESD protection.

The illustration below shows the time tag relation to 1PPS wave form:



The pulse is about 8 microseconds wide, with rise and fall times of about 100 ns. Resolution is approximately 40 ns, where the 40 ns resolution means that the PPS shifting mechanism in the receiver can align the PPS to UTC/GPS time only within +/- 20 ns, but the following external factor limits accuracy to approximately ±1 microsecond:

- Antenna cable length

  Each meter of cable adds a delay of about 2 ns to satellite signals, and a corresponding delay in the 1PPS pulse.

ASCII time tag

Each time tag is output about 0.5 second before the corresponding pulse. Time tags are in ASCII format on a user-selected serial port. The format of a time tag is:

UTC yy.mm.dd hh:mm:ss ab

Where:

- UTC is fixed text.
- *yy.mm.dd* is the year, month, and date.
- *hh:mm:ss* is the hour (on a 24-hour clock), minute, and second. The time is in UTC, not GPS.
- *a* is an integer number representing the position-fix type:

  1 = time solution only

  2 = 1D position and time solution

  3 = currently unused

  4 = 2D position and time solution

  5 = 3D position and time solution
- *b* is the number of GNSS satellites being tracked. If the receiver is tracking 9 or more satellites, b will always be displayed as 9.
- Each time tag is terminated by a *carriage return*, *line feed* sequence. A typical printout looks like:

  UTC 02.12.21 20:21:16 56

  UTC 02.12.21 20:21:17 56

  UTC 02.12.21 20:21:18 56

NOTE – If the receiver is not tracking satellites, the time tag is based on the receiver clock. In this case, a and b are represented by "??". The time readings from the receiver clock are less accurate than time readings determined from the satellite signals.

9

GSOF Message Parsing and Decoding

This topic describes a simple General Serial Output Format (GSOF) message protocol parser. The console utility is written in "C" and compiled in a Linux environment using the GNU Compiler Collection (GCC) version 4.6.3 20120306. The Code has been compiled and validated using Fedora Version 16 (64 Bit) running on VMware Workstation (Virtual Machine) version 9.0.1 build-894247. The source is being provided to Trimble customers who wish to decode and use the GSOF Protocol.

Useful links:

- Refer to the Chapter 9, Output Messages, in the appropriate Trimble BD9xx GNSS receiver manual
- C source file

Source code description

The data is assumed to be the raw GSOF output from a Trimble receiver. That is, it consists of Trimcomm packets (02..03) of type 0x40 in which are embedded GSOF subtype records. The program accepts such data on standard input (either live as part of a '|'-pipeline, or from a file via '<'-redirection. It synchronizes with the individual Trimcomm packets and extracts the contents. When a complete set of GSOF-0x40 packets is collected, the total contents is parsed and listed. For some GSOF subtypes there is a full decoder below and the contents are listed, item by item. Other packets are listed just as Hex bytes. You can write additional routines to the decoder if required, using the routines as models to implement for additional GSOF subtypes.

The program starts with main which collects Trimcomm packets. It then moves to postGsofData() which collects the GSOF data from multiple packets and decides when a complete set has been received. Then it goes to processGsofData() which steps through the collected data parsing the individual GSOF subtype records. If the GSOF subtype is one of the special ones where it has a decoder, that decoder is called, otherwise the program just dumps the Hex bytes of the record. The program runs until the Stdinput indicates end of file (EOF) [see gc()] or the user stops it with a "Ctrl "C" action.

NOTE – This program is not designed to handle corrupted data. It does not contain sophisticated logic to handle corrupted data packets. This source is being provided "As Is". The program was written to enable viewing the contents of well-formed GSOF data, not to debug the overall formatting. There should be some resistance to additional data such as NMEA being mixed into the GSOF stream, as this has not been validated.

Header, defines, types, and routines

This section contains header, defines, types, and routines parsers for individual GSOF records:

Header and defines

```
#include <stdio.h>

#include <unistd.h>

#include <string.h>

#define PI (3.14159265358979)
```

Types

```
typedef unsigned long U32 ;
typedef unsigned short U16 ;
typedef signed short S16 ;
typedef unsigned char U8 ;
```

Global variables

```
/* A few global variables needed for collecting full GSOF packets from multiple Trimcomm packets. */
unsigned char gsofData[2048] ;
int gsofDataIndex ;
```

Function: GetU32

```
/**********************************************************************/
unsigned long getU32( unsigned char * * ppData )
/**********************************************************************/
// Used by the decoding routines to grab 4 bytes and pack them into
// a U32. Fed ppData which is a pointer to a pointer to the start of
// the data bytes. The pointer variable referenced by ppData is moved
// beyond the four bytes.
// This is designed to work on little-endian processors (Like Pentiums).
// Effectively that means we reverse the order of the bytes.
// This would need to be rewritten to work on big-endian PowerPCs.
{
unsigned long retValue ;
unsigned char * pBytes ;
pBytes = (unsigned char *)(&retValue) + 3 ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes = *(*ppData)++ ;
return retValue ;
} /* end of getU32() */
```

Function: GetFloat

```
/*****************************************************************************/
float getFloat( unsigned char * * ppData )
/*****************************************************************************/
// Used by the decoding routines to grab 4 bytes and pack them into
// a Float. Fed ppData which is a pointer to a pointer to the start of
// the data bytes. The pointer variable referenced by ppData is moved
// beyond the four bytes.
// This is designed to work on little-endian processors (Like Pentiums).
// Effectively that means we reverse the order of the bytes.
// This would need to be rewritten to work on big-endian PowerPCs.
{
float retValue ;
unsigned char * pBytes ;
pBytes = (unsigned char *)(&retValue) + 3 ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes = *(*ppData)++ ;
return retValue ;
} /* end of getFloat() */
```

Function: GetDouble

```
/***********************************************************************/
double getDouble( unsigned char * * ppData )
/***********************************************************************/
// Used by the decoding routines to grab 8 bytes and pack them into
// a Double. Fed ppData which is a pointer to a pointer to the start of
// the data bytes. The pointer variable referenced by ppData is moved
// beyond the four bytes.
// This is designed to work on little-endian processors (Like Pentiums).
// Effectively that means we reverse the order of the bytes.
// This would need to be rewritten to work on big-endian PowerPCs.
{
double retValue ;
unsigned char * pBytes ;
pBytes = (unsigned char *)(&retValue) + 7 ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes-- = *(*ppData)++ ;
*pBytes = *(*ppData)++ ;
return retValue ;
} /* end of getDouble() */
```

Function: GetU16

```
/*********************************************************************/
unsigned short getU16( unsigned char * * ppData )
/*********************************************************************/
// Used by the decoding routines to grab 2 bytes and pack them into
// a U16. Fed ppData which is a pointer to a pointer to the start of
// the data bytes. The pointer variable referenced by ppData is moved
// beyond the four bytes.
// This is designed to work on little-endian processors (Like Pentiums).
// Effectively that means we reverse the order of the bytes.
// This would need to be rewritten to work on big-endian PowerPCs.
{
unsigned short retValue ;
unsigned char * pBytes ;
pBytes = (unsigned char *)(&retValue) + 1 ;
*pBytes-- = *(*ppData)++ ;
*pBytes = *(*ppData)++ ;
return retValue ;
} /* end of getU16() */
/**********************************************************************
* The next section contains routines which are parsers for individual
* GSOF records. They are all passed a length (which is listed but
* usually not used) and a pointer to the data bytes that make up the
* record.
*********************************************************************/
```

Position Time

```
/***********************************************************************/
void processPositionTime( int length, unsigned char *pData )
/***********************************************************************/
{
unsigned long msecs ;
unsigned short weekNumber ;
int nSVs ;
int flags1 ;
int flags2 ;
int initNumber ;
printf( " GsofType:1 - PositionTime len:%d\n", length) ;
msecs = getU32( &pData ) ;
weekNumber = getU16( &pData ) ;
nSVs = *pData++ ;
flags1 = *pData++ ;
flags2 = *pData++ ;
initNumber = *pData++ ;
printf( " Milliseconds:%ld Week:%d #Svs:%d "
"flags:%02X:%02X init:%d\n",
msecs,
weekNumber,
nSVs,
flags1,
flags2,
initNumber
) ;
} /* end of processPositionTime() */
```

Latitude, Longitude and Height

```
/*****************************************************************************/
void processLatLonHeight( int length, unsigned char *pData )
/*****************************************************************************/
{
double lat, lon, height ;
printf( " GsofType:2 - LatLongHeight len:%d\n", length ) ;
lat = getDouble( &pData ) * 180.0 / PI ;
lon = getDouble( &pData ) * 180.0 / PI ;
height = getDouble( &pData ) ;
printf( " Lat:%.7f Lon:%.7f Height:%.3f\n",
lat,
lon,
height
) ;
} /* end of processLatLonHeight() */
```

Earth-Centered, Earth-Fixed Position

```
/*****************************************************************************/
void processECEF( int length, unsigned char *pData )
/*****************************************************************************/
{
double X, Y, Z ;
printf( " GsofType:3 - ECEF len:%d\n", length ) ;
X = getDouble( &pData ) ;
Y = getDouble( &pData ) ;
Z = getDouble( &pData ) ;
printf( " X:%.3f Y:%.3f Z:%.3f\n", X, Y, Z ) ;
} /* end of processECEF() */
```

Earth-Centered, Earth-Fixed Delta Position

```
/***********************************************************************/
void processEcefDelta( int length, unsigned char *pData )
/***********************************************************************/
{
double X, Y, Z ;
printf( " GsofType:6 - ECEF Delta len:%d\n", length) ;
X = getDouble( &pData ) ;
Y = getDouble( &pData ) ;
Z = getDouble( &pData ) ;
printf( " X:%.3f Y:%.3f Z:%.3f\n", X, Y, Z ) ;
} /* end of processEcefDelta() */
```

Tangent Plane Delta

```
/***********************************************************************/
void processTangentPlaneDelta( int length, unsigned char *pData )
/***********************************************************************/
{
double E, N, U ;
printf( " GsofType:7 - Tangent Plane Delta len:%d\n", length) ;
E = getDouble( &pData ) ;
N = getDouble( &pData ) ;
U = getDouble( &pData ) ;
printf( " East:%.3f North:%.3f Up:%.3f\n", E, N, U ) ;
} /* end of processTangentPlaneDelta() */
```

Velocity Data

```
/*********************************************************************/
void processVelocityData( int length, unsigned char *pData )
/*********************************************************************/
{
int flags ;
float velocity ;
float heading ;
float vertical ;
printf( " GsofType:8 - Velocity Data len:%d\n", length) ;
flags = *pData++ ;
velocity = getFloat( &pData ) ;
heading = getFloat( &pData ) * 180.0 / PI ;
vertical = getFloat( &pData ) ;
printf( " Flags:%02X velocity:%.3f heading:%.3f vertical:%.3f\n",
flags,
velocity,
heading,
vertical
) ;
} /* end of processVelocityData() */
```

Current UTC Time

```
/*********************************************************************/
void processUtcTime( int length, unsigned char *pData )
/*********************************************************************/
{
printf( " GsofType:16 - UTC Time Info len:%d\n", length) ;
U32 msecs = getU32( &pData ) ;
U16 weekNumber = getU16( &pData ) ;
S16 utcOffset = getU16( &pData ) ;
```

```
U8 flags = *pData++ ;
printf( " ms:%lu week:%u utcOff:%d flags:%02x\n",
msecs,
weekNumber,
utcOffset,
flags
) ;
} /* end of processUtcTime() */
```

PDOP Info

```
/***********************************************************************/
void processPdopInfo( int length, unsigned char *pData )
/***********************************************************************/
{
float pdop ;
float hdop ;
float vdop ;
float tdop ;
printf( " GsofType:9 - PDOP Info len:%d\n", length) ;
pdop = getFloat( &pData ) ;
hdop = getFloat( &pData ) ;
vdop = getFloat( &pData ) ;
tdop = getFloat( &pData ) ;
printf( " PDOP:%.1f HDOP:%.1f VDOP:%.1f TDOP:%.1f\n",
pdop,
hdop,
vdop,
tdop
) ;
} /* end of processPdopInfo() */
```

SV Brief Info

```
/**********************************************************************/
void processBriefSVInfo( int length, unsigned char *pData )
/**********************************************************************/
{
int nSVs ;
int i ;
printf( " GsofType:13 - SV Brief Info len:%d\n", length) ;
nSVs = *pData++ ;
printf( " SvCount:%d\n", nSVs ) ;
for ( i = 0 ; i < nSVs ; ++i )
{
int prn ;
int flags1 ;
int flags2 ;
prn = *pData++ ;
flags1 = *pData++ ;
flags2 = *pData++ ;
printf( " Prn:%-2d flags:%02X:%02X\n", prn, flags1, flags2 );
}
} /* end of processBriefSVInfo */
```

SV Brief Info (All Satellite Systems)

```
/**********************************************************************/
void processAllBriefSVInfo( int length, unsigned char *pData )
/**********************************************************************/
{
int nSVs ;
int i ;
printf( " GsofType:33 - All SV Brief Info len:%d\n", length) ;
nSVs = *pData++ ;
```

```
printf( " SvCount:%d\n", nSVs ) ;
for ( i = 0 ; i < nSVs ; ++i )
{
int prn ;
int system ;
int flags1 ;
int flags2 ;
prn = *pData++ ;
system = *pData++;
flags1 = *pData++ ;
flags2 = *pData++ ;
printf( " %s SV:%-2d flags:%02X:%02X\n",
system == 0 ? "GPS"
: system == 1 ? "SBAS"
: system == 2 ? "GLONASS"
: system == 3 ? "GALILEO"
: system == 4 ? "QZSS"
: system == 5 ? "BEIDOU"
: system == 6 ? "RESERVED" : "RESERVED",
prn, flags1, flags2 );
}
} /* end of processAllBriefSVInfo */
```

SV Detailed Info (All Satellite Systems)

```
/***********************************************************************/
void processAllDetailedSVInfo( int length, unsigned char *pData )
/***********************************************************************/
{
int nSVs ;
int i ;
printf( " GsofType:34 - All SV Detailed Info len:%d\n", length) ;
```

```
nSVs = *pData++ ;
printf( " SvCount:%d\n", nSVs ) ;
for ( i = 0 ; i < nSVs ; ++i )
{
int prn ;
int system ;
int flags1 ;
int flags2 ;
int elevation ;
int azimuth ;
int snr[ 3 ];
prn = *pData++ ;
system = *pData++;
flags1 = *pData++ ;
flags2 = *pData++ ;
elevation = *pData++ ;
azimuth = getU16( &pData ) ;
snr[ 0 ] = *pData++;
snr[ 1 ] = *pData++;
snr[ 2 ] = *pData++;
printf( " %s SV:%-2d flags:%02X:%02X\n"
" El:%2d Az:%3d\n"
" SNR %3s %5.2f\n"
" SNR %3s %5.2f\n"
" SNR %3s %5.2f\n",
system == 0 ? "GPS"
: system == 1 ? "SBAS"
: system == 2 ? "GLONASS"
: system == 3 ? "GALILEO"
: system == 4 ? "QZSS"
: system == 5 ? "BEIDOU"
```

```
: system == 6 ? "RESERVED" : "RESERVED",
prn, flags1, flags2,
elevation, azimuth,
system == 3 ? "E1 " : "L1 ", (float)snr[ 0 ] / 4.0,
system == 3 ? "N/A " : "L2 ", (float)snr[ 1 ] / 4.0,
system == 3 ? "E5 "
: system == 2 ? "G1P" : "L5 ", (float)snr[ 2 ] / 4.0
);
}
} /* end of processAllDetailedSVInfo */
```

SV Detailed Info

```
/**********************************************************************/
void processSvDetailedInfo( int length, unsigned char *pData )
/**********************************************************************/
{
int nSVs ;
int i ;
printf( " GsofType:14 - SV Detailed Info len:%d\n", length) ;
nSVs = *pData++ ;
printf( " SvCount:%d\n", nSVs ) ;
for ( i = 0 ; i < nSVs ; ++i )
{
int prn ;
int flags1 ;
int flags2 ;
int elevation ;
int azimuth ;
int l1Snr ;
int l2Snr ;
prn = *pData++ ;
```

```
flags1 = *pData++ ;
flags2 = *pData++ ;
elevation = *pData++ ;
azimuth = getU16( &pData ) ;
l1Snr = *pData++ ;
l2Snr = *pData++ ;
printf( " Prn:%-2d flags:%02X:%02X elv:%-2d azm:%-3d "
"L1snr:%-5.2f L2snr:%-5.2f\n",
prn,
flags1,
flags2,
elevation,
azimuth,
((double)l1Snr) / 4.0 ,
((double)l2Snr) / 4.0
) ;
}
} /* end of processSvDetailedInfo() */
```

Attitude Info

```
/*************************************************************************/
void processAttitudeInfo( int length , unsigned char *pData )
/*************************************************************************/
{
double gpsTime ;
unsigned char flags ;
unsigned char nSVs ;
unsigned char mode ;
double pitch ;
double yaw ;
double roll ;
```

```
double range ;
double pdop ;
printf( " GsofType:27 - AttitudeInfo len:%d\n",
length
) ;
gpsTime = (double)getU32( &pData ) / 1000.0 ;
flags = *pData++ ;
nSVs = *pData++ ;
mode = *pData++ ;
++pData ; // reserved
pitch = getDouble( &pData ) / PI * 180.0 ;
yaw = getDouble( &pData ) / PI * 180.0 ;
roll = getDouble( &pData ) / PI * 180.0 ;
range = getDouble( &pData ) ;
pdop = (double)getU16( &pData ) / 10.0 ;
printf( " Time:%.3f"
" flags:%02X"
" nSVs:%d"
" mode:%d\n"
" pitch:%.3f"
" yaw:%.3f"
" roll:%.3f"
" range:%.3f"
" pdop:%.1f"
"\n",
gpsTime,
flags,
nSVs,
mode,
pitch,
yaw,
```

```
roll,
range,
pdop
) ;
// Detect if the extended record information is present
if ( length > 42 )
{
float pitch_var ;
float yaw_var ;
float roll_var ;
float pitch_yaw_covar ;
float pitch_roll_covar ;
float yaw_roll_covar ;
float range_var;
// The variances are in units of radians^2
pitch_var = getFloat( &pData ) ;
yaw_var = getFloat( &pData ) ;
roll_var = getFloat( &pData ) ;
// The covariances are in units of radians^2
pitch_yaw_covar = getFloat( &pData ) ;
pitch_roll_covar = getFloat( &pData ) ;
yaw_roll_covar = getFloat( &pData ) ;
// The range variance is in units of m^2
range_var = getFloat( &pData ) ;printf( " variance (radians^2)"
" pitch:%.4e"
" yaw:%.4e"
" roll:%.4e"
"\n",
pitch_var,
yaw_var,
roll_var ) ;
```

```
printf( " covariance (radians^2)"
" pitch-yaw:%.4e"
" pitch-roll:%.4e"
" yaw-roll:%.4e"
"\n",
pitch_yaw_covar,
pitch_roll_covar,
yaw_roll_covar ) ;
printf( " variance (m^2)"
" range: %.4e"
"\n",
range_var ) ;
}
} /* end of processAttitudeInfo() */
```

L-Band Status Info

```
/***********************************************************************/
void processLbandStatus( int length , unsigned char *pData )
/***********************************************************************/
{
unsigned char name[5];
float freq;
unsigned short bit_rate;
float snr;
unsigned char hp_xp_subscribed_engine;
unsigned char hp_xp_library_mode;
unsigned char vbs_library_mode;
unsigned char beam_mode;
unsigned char omnistar_motion;
float horiz_prec_thresh;
float vert_prec_thresh;
```

```
unsigned char nmea_encryption;
float iq_ratio;
float est_ber;
unsigned long total_uw;
unsigned long total_bad_uw;
unsigned long total_bad_uw_bits;
unsigned long total_viterbi;
unsigned long total_bad_viterbi;
unsigned long total_bad_messages;
unsigned char meas_freq_is_valid = -1;
double meas_freq = 0.0;
printf( " GsofType:40 - LBAND status len:%d\n",
length
) ;
memcpy( name, pData, 5 );
pData += 5;
freq = getFloat( &pData );
bit_rate = getU16( &pData );
snr = getFloat( &pData );
hp_xp_subscribed_engine = *pData++;
hp_xp_library_mode = *pData++;
vbs_library_mode = *pData++;
beam_mode = *pData++;
omnistar_motion = *pData++;
horiz_prec_thresh = getFloat( &pData );
vert_prec_thresh = getFloat( &pData );
nmea_encryption = *pData++;
iq_ratio = getFloat( &pData );
est_ber = getFloat( &pData );
total_uw = getU32( &pData );
total_bad_uw = getU32( &pData );
```

```
total_bad_uw_bits = getU32( &pData );
total_viterbi = getU32( &pData );
total_bad_viterbi = getU32( &pData );
total_bad_messages = getU32( &pData );
if( length > 61 )
{
meas_freq_is_valid = *pData++;
meas_freq = getDouble( &pData );
}
printf( " Name:%s"
" Freq:%g"
" bit rate:%d"
" SNR:%g"
"\n"
" HP/XP engine:%d"
" HP/XP mode:%d"
" VBS mode:%d"
"\n"
" Beam mode:%d"
" Omnistar Motion:%d"
"\n"
" Horiz prec. thresh.:%g"
" Vert prec. thresh.:%g"
"\n"
" NMEA encryp.:%d"
" I/Q ratio:%g"
" Estimated BER:%g"
"\n"
" Total unique words(UW):%d"
" Bad UW:%d"
" Bad UW bits:%d"
```

```
"\n"
" Total Viterbi:%d"
" Corrected Viterbi:%d"
" Bad messages:%d"
"\n"
" Meas freq valid?:%d"
" Meas freq:%.3f"
"\n"
,
name,
freq,
bit_rate,
snr,
hp_xp_subscribed_engine,
hp_xp_library_mode,
vbs_library_mode,
beam_mode,
omnistar_motion,
horiz_prec_thresh,
vert_prec_thresh,
nmea_encryption,
iq_ratio,
est_ber,
total_uw,
total_bad_uw,
total_bad_uw_bits,
total_viterbi,
total_bad_viterbi,
total_bad_messages,
meas_freq_is_valid,
meas_freq
```

```
) ;
} /* end of processLbandStatus() */
```

Function: Process GSOF Data

```
/***********************************************************************/
void processGsofData( void )
/***********************************************************************/
/* Called when a complete set of GSOF packets has been received.
* The data bytes collected are available in global gsofData and the
* number of those bytes is in gsofDataIndex.
*
* This routine just goes through the bytes and parses the sub-type
* records. Each of those has a Type and a Length. If the type is
* one of the special types we know about, we call the proper parser.
* Otherwise we just hex-dump the record.
*/
{
int i ;
int gsofType ;
int gsofLength ;
unsigned char * pData ;
printf( "\nGSOF Records\n" ) ;
pData = gsofData ;
while (pData < gsofData + gsofDataIndex )
{
gsofType = *pData++ ;
gsofLength = *pData++ ;
// If the type is one that we know about, then call the specific
// parser for that type.
if ( gsofType == 1 )
{
```

```
processPositionTime( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 2 )
{
processLatLonHeight( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 3 )
{
processECEF( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 4 )
{
processLocalDatum( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 8 )
{
processVelocityData( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 9 )
{
processPdopInfo( gsofLength, pData ) ;
```

```
pData += gsofLength ;
}
else
if ( gsofType == 13 )
{
processBriefSVInfo( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 16 )
{
processUtcTime( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 33 )
{
processAllBriefSVInfo( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 34 )
{
processAllDetailedSVInfo( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 14 )
{
processSvDetailedInfo( gsofLength, pData ) ;
pData += gsofLength ;
```

```
}
else
if ( gsofType == 27 )
{
processAttitudeInfo( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 26 )
{
processPositionTimeUtc( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 6 )
{
processEcefDelta( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 7 )
{
processTangentPlaneDelta( gsofLength, pData ) ;
pData += gsofLength ;
}
else
if ( gsofType == 40 )
{
processLbandStatus( gsofLength, pData ) ;
pData += gsofLength ;
}
```

```
else
{
// Not a type we know about. Hex dump the bytes and move on.
printf( " GsofType:%d len:%d\n ",
gsofType,
gsofLength
) ;
for ( i = 0 ; i < gsofLength ; ++i )
{
printf( "%02X%s",
*pData++,
i % 16 == 15 ? "\n " : " "
) ;
}
// Terminate the last line if needed.
if (gsofLength %16 != 0)
printf( "\n" ) ;
}
printf( "\n" ) ;
}
printf( "\n" ) ;
} /* end of processGsofData() */
```

Function: Post GSOF Data

```
/***********************************************************************/
void postGsofData( unsigned char * pData, int length )
/***********************************************************************/
// Called whenever we get a new Trimcomm GSOF packet (type 0x40).
// These all contain a portion (or all) of a complete GSOF packet.
// Each portion contains a Transmission Number, an incrementing value
// linking related portions.
```

```
// Each portion contains a Page Index, 0..N, which increments for each
// portion in the full GSOF packet.
// Each portion contains a Max Page Index, N, which is the same for all
// portions.
//
// Each portion's data is appended to the global buffer, gsofData[].
// The next available index in that buffer is always gsofDataIndex.
// When we receive a portion with Page Index == 0, that signals the
// beginning of a new GSOF packet and we restart the gsofDataIndex at
// zero.
//
// When we receive a portion where Page Index == Max Page Index, then
// we have received the complete GSOF packet and can decode it.
{
int gsofTransmissionNumber ;
int gsofPageIndex ;
int gsofMaxPageIndex ;
int i ;
gsofTransmissionNumber = *pData++ ;
gsofPageIndex = *pData++ ;
gsofMaxPageIndex = *pData++ ;
printf( " GSOF packet: Trans#:%d Page:%d MaxPage:%d\n",
gsofTransmissionNumber,
gsofPageIndex,
gsofMaxPageIndex
) ;
// If this is the first portion, restart the buffering system.
if (gsofPageIndex == 0)
gsofDataIndex = 0 ;
// Transfer the data bytes in this portion to the global buffer.
for (i = 3 ; i < length ; ++i)
```

```
gsofData[ gsofDataIndex++ ] = *pData++ ;
// If this is the last portion in a packet, process the whole packet.
if (gsofPageIndex == gsofMaxPageIndex)
processGsofData() ;
} /* end of postGsofData() */
```

Function: Get Character (gc)

```
/****************************************************************************/
int gc( void )
/****************************************************************************/
/* This is a getchar() wrapper. It just returns the characters
* from standard input. If it detects end of file, it aborts
* the entire program.
*
* NOTE: This function is not optimal because if the program is in the middle of a packet
there is
* no indication. This is a simple parsing application
*/
{
int c ;
c = getchar() ;
if (c != EOF)
return c ;
printf( "END OF FILE \n" ) ;
_exit( 0 ) ;
} /* end of gc() */
```

Function: Main

```
/****************************************************************************/
int main( int argn, char **argc )
/****************************************************************************/
/* Main entry point. Looks for Trimcomm packets. When we find one with
```

```
* type 0x40, its bytes are extracted and passed on to the GSOF
* handler.
*/
{
int tcStx ;
int tcStat ;
int tcType ;
int tcLength ;
int tcCsum ;
int tcEtx ;
unsigned char tcData[256] ;
int i ;
printf( "GSOF Parser\n") ;
while ( 1 )
{
tcStx = gc() ;
if ( tcStx == 0x02 )
{
tcStat = gc() ;
tcType = gc() ;
tcLength = gc() ;
for ( i = 0 ; i < tcLength ; ++i )
tcData[i] = gc() ;
tcCsum = gc() ;
tcEtx = gc() ;
printf( "STX:%02Xh Stat:%02Xh Type:%02Xh "
"Len:%d CS:%02Xh ETX:%02Xh\n",
tcStx,
tcStat,
tcType,
tcLength,
```

```
tcCsum,
tcEtx
) ;
if (tcType == 0x40)
postGsofData( tcData, tcLength ) ;
}
else
printf( "Skipping %02X\n", tcStx ) ;
}
return 0 ;
} // main
```