Beagle High Speed USB 480 Protocol Analyzer Features

- Non-intrusive high-, full-, and low-speed monitoring
- Monitor packetsin real-time as theyappear on thebus
- Large64MB onboard hardwarebuffer
- Digital inputs and outputsfor synchronizing withexternal logic
- Repetitive packet compression
- Packet-level timing with 16.6 ns resolution
- Linux and Windows compatible

Beagle USB 12 Protocol Analyzer Features

- Non-intrusive full-, low-speed monitoring (12 and 1.5 Mbps)
- Monitor packetsin real-time as theyappear on thebus
- Repetitive packet compression
- Bit-level timing with 21 ns resolution
- High-speed USB uplink to analysis computer
- Linux and Windows compatible

BeagleI^{2}C/SPI/MDIOProtocol Analyzer Features

- Non-intrusiveI^{2}Cmonitoringupto4MHz
- Non-intrusive SPI monitoring up to 24 MHz
- Non-intrusive MDIO monitoring up to 2.5 MHz
- Monitor packetsin real-time as theyappear on thebus.
- User selectable bit-level timing (up to 20 ns resolution)
- High-speed USB uplink to analysis computer
- Linux and Windows compatible

Summary

The Beagle™ Protocol Analyzers are non-intrusive debugging tools. Developers canwatch datain real-time as they occur. The datais appropriately parsedfor the protocolof interest. Like allTotal Phase products, the Beagle analyzerisa low-cost, cross-platformdevicefor Windows and Linux.

TOTAL PHASE

Beagle

Protocol Analyzers

Data Sheet v3.02January28, 2008

1 General Overview

1.1 USB Background

USB History

UniversalSerialBus(USB)isa standard interfacefor connectingperipheraldevicestoahost computer. The USB systemwas originallydevisedby agroupof companies including Compaq, Digital Equipment, IBM, Intel, Microsoft, and Northern Telecom to replace the existing mixed connector system with a simpler architecture.

USB was designed to replace the multitude of cables and connectors required to connect pe-ripheral devices to a host computer. The main goal of USB was to make the addition of pe-ripheral devices quickand easy. All USB devices share somekeycharacteristics to make this possible.AllUSBdevicesare self-identifyingonthebus.Alldevicesare hot-pluggabletoallow for true Plug’n’Playcapability. Additionally, some devices can draw power from the USB which eliminates the needforextrapower adapters.

To ensure maximum interoperability the USB standard de□nes all aspects of the USB system fromthephysicallayer (mechanicalandelectrical)allthewayuptothesoftwarelayer.TheUSB standard is maintained and enforcedbythe USB Implementer’sForum (USB-IF). USB devices must pass a USB-IF compliance test in order to be considered in compliance and to be able to use the USB logo.

The USB standard speci□es different □avors of USB: low-speed, full-speed and high-speed. USB-IF has also released additional specs that expand the breadth of USB. These are On-The-Go (OTG) and Wireless USB. Although beyond the scope of this document, details on these specs canbefound on the USB-IFwebsite.

Thekeydifference betweenlow, full, and high speedis bandwidth.

Low	1.5 Mbps
Full	12 Mbps
High	480 Mbps

The USB speci□cation can be viewed and downloaded on the USB-IF website.

Architectural Overview

USBisa host-scheduled, token-based serialbus protocol. USB allowsfor the connectionofup to 127 devices on a single USB host controller. A host PC can have multiple host controllers which increases the maximum number of USB devices that can be connected to a single com-puter.

Devices can be connected and disconnected at will. The host PC is responsiblefor installing and uninstallingdriversfor the USBdevices on an as-needed basis.

Asingle USB system comprises of a USB host and one or more USB devices. There can also bezero or more USB hubsin the system.AUSB hubis special classofdevice. The hub allows the connection of multiple downstream devices to an upstream host or hub. In this way, the number of devices that can be physically connected to a computer can be increased.

AUSB device is a peripheral device that connects to the host PC. The range of functionality of USB devices is ever increasing. The device can support either one function or manyfunctions. For example a single multi-function printer maypresent several devices to the host when it is connected via USB.It can presentaprinterdevice,a scannerdevice,afaxdevice, etc.

All the devices on a single USB must share the bandwidth that is available on thebus. It is possiblefora hostPCtohavemultiplebuses whichwouldallhave theirown separate band-width. Most often, the ports on most motherboards are paired, such that each bus has two downstream ports.

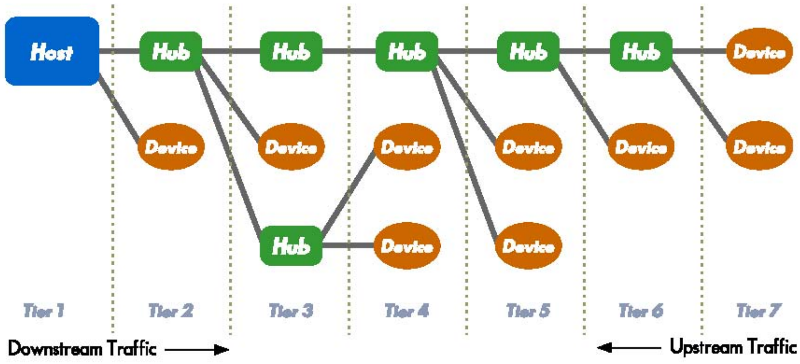

Figure1:Sample USB BusTopology.AUSB can only have a single USB host device. This host can support up to 127 different devices on asingleport. Thereisan upper limitof7tiersofdevices which means thata maximumof5hubs canbeconnected inline.

The USB has a tiered star topology (Figure 1). At the root tier is the USB host. All devices connect to the host either directly or via a hub. According to the USB spec, a USB host can only support a maximum of seven tiers.

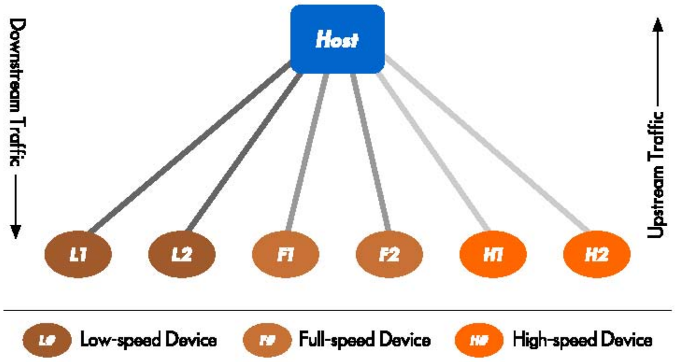

Figure 2: USB Broadcast Ahost broadcasts information to all the devices below it. Low-speed and high-speed enabled devices will only see traf□c at their respective speeds. Full-speed devices can see both their speed and low-speed traf□c.

USB works through a unidirectional broadcast system. When a host sends a packet, all down

stream devices will see that traf□c. If the host wishes to communicate with a speci□c device, it must include the address of the device in the token packet. Upstream traf□c (the response from devices) are only seenbythe host or hubs that are directly on the return path to the host.

Thereare,however, afewcaveatswhen dealingwithdevicesthatareofdifferentspeeds.Low-speed and high-speed devices are isolated from traf□c at speeds other then their own. They will only see traf□c that is at their respective speeds. Referring to Figure 2, this means that downstreamtraf□ctodeviceH1 willbe seenbydeviceH2(and viceversa). Also,downstream traf□ctodeviceL1 willbe seenbyL2(and viceversa). However, full-speeddevices can see traf□c at its own speed, as well as low-speed traf□c, using a special signaling mode dubbed low-speed-over-full-speed. This means that downstream traf□c to F1 will be seenbyF2 (and vice versa) with standard full-speed signaling, and downstream traf□c to either L1 or L2 will also be seenbybothF1 andF2 through the speciallow-speed-over-full-speed signaling.

Theoryof Operations

This introduction is a general summary of the USB spec. Total Phase strongly recommends that developers consult the USB speci□cation on the USB-IF website for detailed and up-to-date information.

USB Connectors

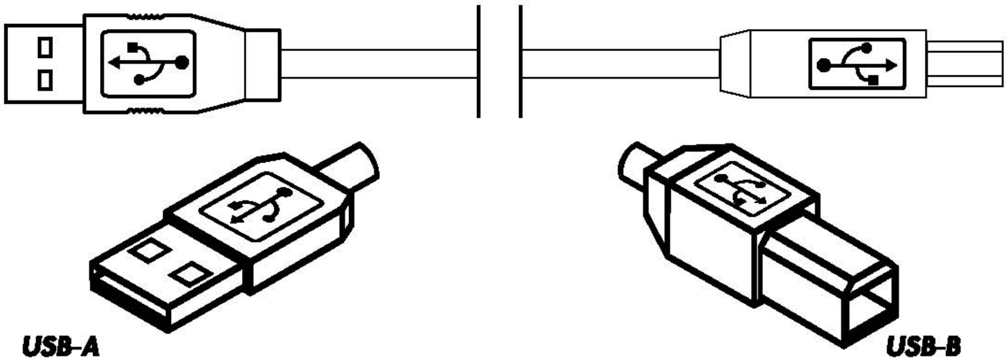

Figure3:USB Cable A USB cable has two different types of connectors: “A” and “B”. “A” connectors connect upstream to-wards the Host and “B” connectors connect downstream to the Devices.

USB cables have two different types of connectors: “A” and “B”. “A” type connectors connect towards the host or upstream direction. “B” connectors connect to downstream devices, though many devices have captive cables eliminating the need for “B” connectors. The “A” and “B” connectorsare de□ nedintheUSBspectopreventloopbacksinthebus.Thispreventsahost from being connected to a host, or conversely a device to a device. It also helps enforce the tieredstar topologyofthebus.USBhubshaveone“B”portandmultiple“A”portswhichmakes it clear which port connects to the host and which to downstream devices.

The USB spec has been expanded to include Mini-A and Mini-B connectors to support small USB devices. The USB On-The-Go (OTG) spec has introduced the Micro-A plug, Micro-B plug

and receptacle, and the Micro-AB receptacle to allowfor device-to-device connections. (The previous Mini-A plug and Mini-AB receptacle have now been deprecated.)

USB Signaling

All USBdevices are connectedby afour wire USB cable. Thesefour lines areV_{BUS}, GND and the twisted pair: D+ and D-. USB uses differential signaling on the two data lines. There are four possible digital line states that thebus canbe in: single-ended zero (SE0), single-ended one(SE1),J,andK.The

single-endedline statesare de□nedthe same regardlessofthespeed. However, the de□nitions of theJandKline states change depending on thebus speed. Their de□nitions are described in Table 1. All data is transmitted through the J and K line states. An SE1 condition should never be seen on the bus, except for allowances during transitions
between the other line states.

Table1:Differential Signal Encodings

Single-ended zero (SE0)	0	0
Single-ended one (SE1)	1	1
Low-speed J	0	1
Low-speed K	1	0

High-/Full-speedJ
High-/Full-speedK

D+

1 0 D

0 1

The actual data on thebusis encoded through the line statesby a nonreturn-to-zero-inverted (NRZI)digitalsignal.InNRZIencoding,adigital1is representedby nochangeinthelinestate anda digital0is representedasa changeofthe line state. Thus,everytimea0istransmitted theline statewillchangefromJtoK,orviceversa. However,ifa1isbeingsenttheline state will remain the same.

USB has no synchronizing clockline between the host anddevice. However, the receiver can resynchronize wheneveravalidtransitionis seenonthebus. Thisis possibleprovided thata transitionintheline stateisguaranteedwithina □ xedperiodoftimedeterminedbytheallowable clockskew betweenthe receiverandtransmitter.To ensure thatatransitionis seenonthebus withinthe requiredtime,USBemploysbit stuf □ ng. After6consecutive1sinadata stream(i.e. notransitions ontheD+andD-linesfor6 clockperiods),a0is insertedtoforceatransition of the line states. This is performed regardless of whether the next bit would have induced a transition or not. The receiver, expecting the bit stuff, automatically removes the0from the data stream.

Bus Speed

Thebus speed determines therate at which bits are sent across thebus. There are currently three speeds at which wired USB operates: low-speed (1.5 Mbps), full-speed (12 Mbps), and high-speed (480 Mbps). In order to determine the bus speed of a full-speed or low-speed device,thehostmustsimplylookattheidle stateofthebus. Full-speeddeviceshaveapull-up resistor on the

D+ line, whereas low-speed devices have a pull-up resistor on the D-line. Therefore, if the D+ line is high when idle, then full-speed connectivity is established. If the D-

line is high when idle, then low-speed connectivity is in effect. A full-speed device does not havetobecapableofrunningatlow-speed,andviceversa. Afull-speedhostorhub,however, must be capable of communicating with both full-speed and low-speed devices.

Withthe introductionof high-speedUSB, high-speed hostsandhubsmustbeableto communi-cate with devices of all speeds. Additionally, high-speed devices must be backward compatible for communication at full-speed with legacy hosts and hubs. To facilitate this, all high-speed hosts and devices initially operate at full-speed and a high-speed handshake must take place before a high-speed capable device and a high-speed capable host can begin operating at high-speed. The handshake begins when a high-speed capable host sees a full-speed device attached. Because high-speeddevicesmust initially operateat full-speed when rst connected, they must pull the D+ line high to identify as a full-speed device. The host will then issue a reset on the bus and wait to see if the device responds with a Chirp K, which identies the device as being high-speed capable. If the host does not receiveaChirpK,it quits the high-speed handshake sequence and continues with normal full-speed operation. However, if the host receives a ChirpK, it responds to the device with alternating pairs of ChirpK's and Chirp J's to tell the device that the host is high-speed capable. Upon recognizing these alternating pairs, the device switches to high-speed operation and disconnects its pull-up resistor on the D+ line. The high-speed connection is now established and both the host and the device begin communicating at high-speed. See the USB specicationfor more details on the high-speed handshake.

To accommodate high-speed data-rates and avoid transceiver confusion, the signaling levels of high-speed communication is much lower than that of full and low-speed devices. Full and low-speeddevices operate witha logical highlevelof3.3V on theD+ andD-lines.For high-speed operation, signaling levels on the D+ and D-lines are reduced to 400 mV. Because the high-speed signaling levels are so low, full and low-speed transceivers are not capable of seeing high-speed trafc.

To accommodate the high-speed signaling levels and speeds, both hosts and devices use ter-mination resistors. In addition, during the high-speed handshake, the device must release its full-speed pull-up resistor. But during the high-speed handshake, often times the host will acti-vate its termination resistors before the device releases its full-speed pull-up resistor. In these situationsthehostmaynotbeabletopulltheD+linebelowthe thresholdlevelofits high-speed receivers. This may cause the host to seea spurious ChirpJ (dubbeda TinyJ) on the lines. Thisisanartifactonthebusduetothevoltage dividereffect betweenthedevice's1.5Kohm pull-up resistor and the host's45 ohm termination resistor. Hosts anddevicesmustbe robust againstthis situation. Oncethedevicehasswitchedto high-speedoperationtheTinyJwillno longer be present, since the device will have released its pull-up resistor.

Endpoints and Pipes

The endpoint is the fundamental unit of communication in USB. All data is transferred through virtual pipes between the host and these endpoints. All communication betweena USB host and a USB device is addressed to a specic endpoint on the device. Each device endpoint is a unidirectional receiver or transmitter of data; either specied as a sender or receiver of data from the host.

A pipe represents a data pathwaybetween the host and the device. A pipe may be unidirec-tional (consisting of only one endpoint) or bidirectional (consisting of two endpoints in opposite

directions).

Aspecial pipe is the Default Control Pipe. It consists of both the input and output endpoints 0. It is required on all devices and must be available immediately after the device is powered. The host uses this pipe to identify the device and its endpoints and to con□gure the device.

Endpoints are not all the same. Endpoints specify their bandwidth requirements and the way that theytransfer data. There arefour transfer typesfor endpoints:

Control

Non-periodictransfers.Typically,usedfordevice con□guration, commands,and statusoperation.

Interrupt

This is a transaction that is guaranteed to occur within a certain time interval. The device will specify the time interval at which the host should checkthe device to see if there is new data. Thisis usedbyinputdevices such as mice andkeyboards.

Isochronous

Periodic and continuous transferfor time-sensitive data. There is no error checking or retrans-mission of the data sent in these packets. This is usedfor devices that need to reserve band-widthandhaveahigh toleranceto errors. Examples includemultimediadevicesfor audioand video.

Bulk

General transfer schemefor large amounts of data. This isfor contexts where it is more im-portantthatthe dataistransmitted without errors thanforthe datato arriveina timely manner. Bulk transfers have the lowest priority. If thebus isbusy with other transfers, this transaction maybe delayed. The data is guaranteed to arrive without error. If an error is detected in the CRCs, the data will be retransmitted. Examples of this type of transfer are □les from a mass storage device or the output from a scanner.

USBPackets

All USB packets are prefacedby a SYNC □eld and thenaPacket Identi□er (PID)byte.Packets are terminated with an End-of-Packet (EOP).

TheSYNC □ eld,whichisa sequenceofKJpairsfollowedby2K'sonthedatalines,servesas aStartofPacket(SOP)markerandisusedto synchronizethedevice'stransceiverwiththatof the host. This SYNC □eldis8bits longfor full/low-speed and32 bits longfor high speed.

The EOP □eld varies depending on the bus speed. For low-or full-speed buses, the EOP consistsofanSE0fortwobit times. For high-speedbuses, becausethebusisatSE0 when it is idle, a different method is used to indicate the end of the packet. For high-speed, the transmitter induces a bit stuff error to indicate the end of the packet. So if the line state before theEOPisJ,thetransmitterwillsend 8-bitsofK.Theexceptiontothisisthe high-speedSOF EOP, in which case the high-speed EOP is extended to 40-bits long. This is done for bus disconnect detection.

ThePIDisthe□rstbyteofvaliddatasent acrossthebus,andit encodesthepackettype.The

PIDmaybefollowedbyanywherefrom0to1026bytes, dependingonthepackettype.ThePID byteis self-checking;in orderforthePIDtobevalid,thelast4bitsmustbeaone's complement ofthe □rst4bits.Ifa receivedPIDfailsitscheck,the remainderofthepacketwillbe ignoredby the USB device.

There arefour typesof PID which are describedinTable 2.

Table2:USBPacketTypes

PID Type	PID Name	Description
Token	OUT IN SOF SETUP	Host to device transfer Device to Host transfer Start of Frame marker Host to device control transfer
Data	DATA0 DATA1 DATA2 MDATA	Data packet Data packet High-Speed Data packet Split/High-Speed Data packet
Handshake	ACK NAK STALL NYET	The data packet was received error free Receiver cannot accept data or the transmitter could not send data Endpoint halted or control pipe request is not sup-ported No response yet
Special	PRE ERR SPLIT PING EXT	Preamble to full-speed hub for low-speed traf □c Error handshake for Split Transaction Preamble to high-speed hub for low/full-speed traf□c High-speed □ow control token Protocol extension token

Theformat of the IN, OUT, and SETUPToken packets is shown in □gure 4. Theformat of the SOFpacketisshownin □gure 5. Theformatof the Data packetsis shownin □gure 6. Lastly, the formatof the Handshake packetsis shownin Figure7.

Figure 4: TokenPacketFormat

DataTransactions

Datatransactions occurin three phases:Token, Data,and Handshake.

SYNC	PID	FRAMENUMBER	CRC	EOP
8 bits (low/full)/32 bits (high)	8 bits	11 bits	5 bits	n/a

Figure 5: Start-Of-Frame (SOF)PacketFormat

SYNC	PID	DATA	CRC	EOP
8 bits (low/full)/32 bits (high)	8 bits	up to 8 bytes (low)/1023 bytes (full)/1024 bytes (high)	16 bits	n/a

Figure 6: DataPacketFormat

SYNC	PID	EOP
8 bits (low/full)/32 bits (high)	8 bits	n/a

Figure 7: HandshakePacketFormat

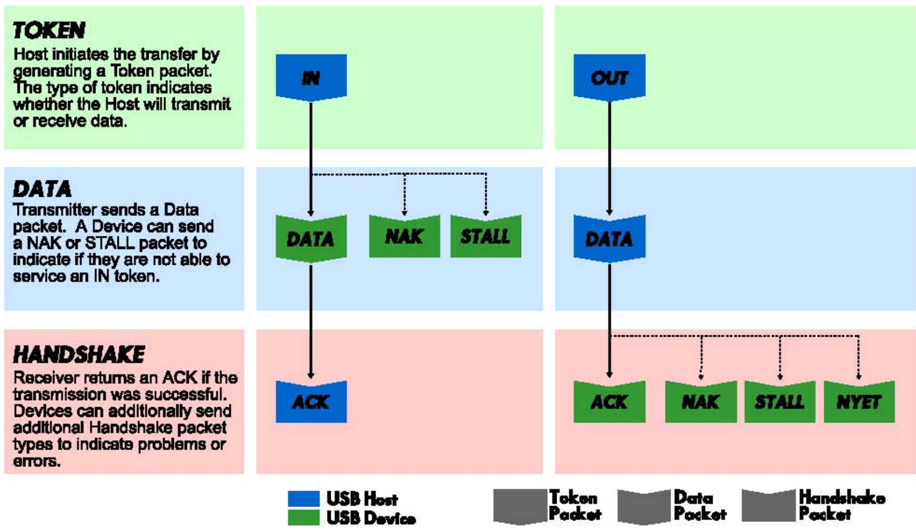

Figure 8: The Three PhasesofaUSBTransfer

All communication on the USBis host-directed. In theToken phase, the host will generatea Token packet which will addressa speci□cdevice/endpoint combination. AToken packet can be IN, OUT, or SETUP.

IN	The host is requesting data from the addressed dev/ep.
OUT	The host is sending data to the addressed dev/ep.
SETUP	The host is transmitting control information to the device.

In the data phase, the transmitter will send one data packet. For INrequests, the device may senda NAKor STALLpacket during the data phase to indicate that it isn't able to service the token that it received.

Finally, in the Handshake phase the receiver can send an ACK, NAK, or STALLindicating the success orfailureof thetransaction.

All of the transfers described above follow this general scheme with the exception of the Isochronous transfer. In this case, no Handshake phase occurs because it is more impor-tantto streamdataoutinatimelyfashion. Itis acceptabletodroppackets occasionallyand thereisno needtowaste timebyattemptingto retransmit thoseparticularpackets.

PollingTransactions

It is possible that when a host requests data or sends data that the device will not be able to service the request. This could occur if the device has no new information to provide the host or is perhapstoobusyto send/receiveanydata.In thesesituationsthedevicewill NAKthe host. If the data transfer is a Control or Bulk transfer, the host will retrythe transaction. However, if it is Isochronous or Interrupt transfer, it will not retrythe transaction.

Ona fullorlow-speedbus,ifthetransactionis repeated,itis repeatedinits entirety. Thisis true regardless of the direction of the data transfer. If the host is requesting information, it will continue to send INtokens until the device sends data. Until then, the device responds with a NAK, leading to the multitude of IN+NAKpairs that are commonly encountered onabus. This does not have much consequence as an INtoken is only3bytes and the NAKis only1byte. However, if the host is transmitting data there is the potential for graver consequences. For example,ifthehost attemptedtosend64bytesofdatatoadevice,butthedevice responded witha NAK, thehost will retrythe entire data transaction. This requires sending the entire 64-byte data payload repeatedly until the device responds with an ACK. This has the potential to waste a signi□cant amount of bandwidth. It isfor this reason that high-speed hosts have an additionalfeature whenthedevice signalsthe inabilityto acceptany more data.

Whena high-speed host receivesa NAKafter transmitting data, instead of retransmitting the en-tiretransaction,it simply sendsa3byte PINGtoken to poll the device and endpoint in question. (Alternatively, if the device responds to the OUT+DATAwitha NYEThandshake, it means that the device acceptedthedatainthe currenttransactionbutisnotreadyto accept additionaldata, and the host should PINGthe device before transmitting more data.) The host will continue to PINGthe device until it responds with an ACK, which indicates to the host that it is ready to receive information. At that point, the host will transmit a packet in its entirety.

HubTransactions

Hubsmakeit possibletoexpandthenumberof possibledevices that canbe attachedtoa single host. There are two types of hubs that are commercially available for wired USB: full-speed hubsand high-speed hubs. Both typesof hubshave mechanismsfor dealing withdownstream devices that are not of their speed.

Full-speed hubs can, at most, transmit at 12 Mbps. This means that all high-speed devices that arepluggedintoafull-speedhubare automaticallydowngradedto full-speeddatarates.Onthe other hand, low-speed devices are not upgraded to full-speed data rates. In order to send data to low-speed devices, the hub must actually pass slower moving data signals to those devices. The host (or high-speed hub) is the one that generates these slower moving signals on the full-speedbus. Ordinarilythelow-speedportsonthehubarequiet.Whenalow-speedpacket needstobe sentdownstream,itisprefacedwitha PREPID. This opens up the low-speed ports. Note that the PREis sentat full-speed datarates,butthefollowingtransactionistransmittedat low-speed data rates.

High-speed hubs only communicate at 480 Mbps with high-speed host. Theydo not downgrade the link between the host and hub to slower speeds. However, high-speed hubs must still deal with slower devices being downstream of them. High-speed hubs do not use the same mechanism as full-speed hubs. There would be a tremendous cost on bandwidth to other high-speeddevicesonthebusiflow-speedor full-speed signalingrateswereusedbetweenthehost and the hub of interest. Thus, in order to save bandwidth, high-speed hosts do not send the PREtokento high-speed hubs,butrathera SPLITtoken. The SPLITtoken is similar to the PREin thatit indicatestoa hub that thefollowingtransactionisfora slower speeddevice,however the datafollowing the SPLITis transmitted to the hub at high-speed data rates and does not choke the high-speedbus.

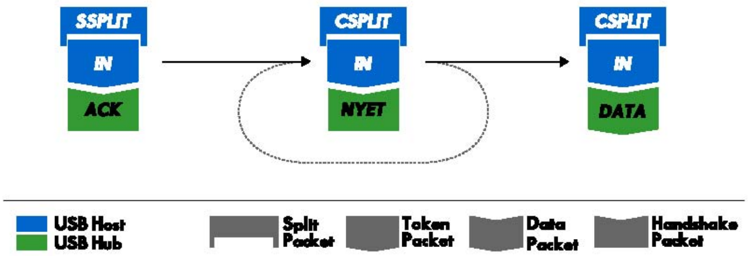

Figure9:Split BulkTransactions When full/low-speed USB traf□c is sent through a high-speed USB hub, the transactions are preceded by a SPLIT token to allow the hub to asynchronously handle the full/low-speed traf□c withoutblocking other high-speedtraf□cfromthe host. Inthisexample,bulkpacketsfora full-speeddevicearebeing sent through the high-speed hub. Multiple CSPLIT+IN+NYET transactions can occur on thebus until the high-speed hub is ready to return theDATAfrom the downstream full/low-speed device.

Although all SPLITtransactionshavethesamePID,therearetwoover-archingtypesof SPLITs: Start SPLITs(SSPLIT)and CompleteSPLITs(CSPLIT). SSPLITs are only used the □rst time that the host wishes to send a given transaction to the device. Following that, it polls the hub for the device's response with CSPLITs. The hub may respond manytimes with NYETbefore supplying the host with the device's response. Once this transaction is complete, it will begin the next hub transaction with an SSPLIT. Figure 9illustratesanexampleofhubtransaction.

Start-of-FrameTransactions

Start-of-Frame(SOF)transactions are issued by the host at precisely timed intervals. These tokens do not cause any device to respond, and can be usedby devicesfor timing reasons. The SOFprovides two pieces of timing information. Because of the precisely timed intervals of SOFs, when a device detects an SOFit knows that the interval time has passed. All SOFs also include a frame number. This is an 11-bit value that is incremented on every new frame.

SOFs are also used tokeep devices from going into suspend. Devices will go into suspend if theysee an idlebusfor anextended periodoftime.By providing SOFs, the host is issuing traf□c on thebus andkeepingdevices from entering their suspended state.

Full-speed hosts will send1 SOFevery millisecond. High-speed hosts divide the frame into8 microframes, and send an SOFat each microframe (i.e., every125 microseconds). However, the high-speed hub will only increment the frame number after an entire frame has passed. Therefore,a high-speed host will repeat the sameframenumber8times before incrementing it.

Low-speed devices are never issued SOFsasitwould requiretoomuch bandwidthonan already slower-speed bus. Instead, to keep low-speed devices from going into suspend, hosts will issueakeep-aliveeverymillisecond. Thesekeep-alives are short SE0events on thebus that lastforapproximately1.33 microseconds.They arenotinterpretedasvaliddata,andhaveno associated PID.

ExtendedTokenTransactions

The new LinkPower Management addendum to the USB 2.0 Speci□cation hasexpanded the number of PIDs through the use of the previously reserved PID, 0xF0. The extended token format is a two phase transaction that begins with a standard token packet that has the EXTPID.Following this packetis theextended token packet, which takesa similarform. It begins with an 8-bit SubPID and ends with a 5-bit CRC, however the 11 remaining bits in the middle will have different meaning depending on the type of SubPID.

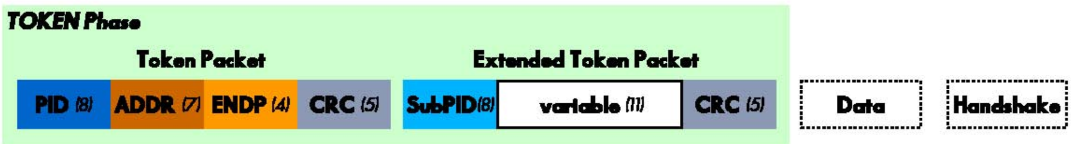

Figure 10: *Extended Token Transaction*
In an extended token transaction, the token phase of the transaction has two token packets. The first packet uses the EXT PID. The content of the second packet will depend on the particular SubPID specification. The subsequent Data and Handshake phases will depend on the value of the SubPID as well.

Following this token phase, the device will respond with the appropriate data or handshake, depending on the protocol associated with that SubPID. Currently, the only de□ned SubPID is for link power management(LPM).For more details, please referto the LinkPower Management addendum.

Enumeration and Descriptors

When a device is plugged into a host PC, the device undergoes Enumeration. This means that the host recognizes the presence of the device and assigns it a unique 7-bit device address. The host PC then queries the devicefor its descriptors, which contains information about the speci□c device. There are various types of descriptors as outlined below.

Device Descriptor: Each USB device can only have a single Device Descriptor. This descriptor

contains information that applies globally to the device, such as serial number,

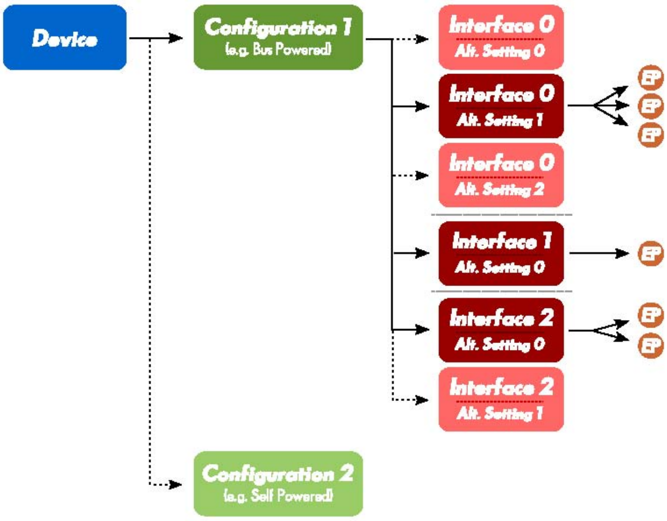

Figure11:USB Descriptors Hierarchyof descriptorsofaUSBdevice.Adevicehasa singleDevice descriptor. TheDevice descriptor canhavemultiple Con□gurationdescriptors,butonlyasingleonecanbeactiveatatime.TheCon□g–uration descriptor can de□ne one or more Interface descriptors. Each of the Interface descriptors can have oneor more alternate settings,butonly one setting canbe activeata time. The Interface descriptor de□nes one or more Endpoints.

vendor ID, product ID, etc. The device descriptor also has information about the device class. The hostPC can use thisinformationtohelp determine whatdriverto loadforthe device.

Con□guration Descriptor: A device descriptor can have one or more con□guration de–scriptors. Eachof these descriptors de□neshow thedeviceispowered(e.g.buspowered or self powered), the maximum power consumption, and what interfaces are available in thisparticularsetup.Thehostcanchoose whethertoread justthecon□gurationdescrip–tor or the entire hierarchy(con□guration, interfaces, and alternate interfaces) at once.

Interface Descriptor:Acon□guration descriptor de□nes one or more interface descriptors. Each interfacenumber canbe subdivided intomultiple alternate interfaces thathelp more □nely modify the characteristics of a device. The host PC selects particular alternate interface depending on what functions it wishes to access. The interface also has class information which the host PC can use to determine what driver to use.

Endpoint Descriptor: An interface descriptor de□nes one or more endpoints. The end–point descriptor is the last leaf in the con□guration hierarchyand it de□nes the bandwidth requirements, transfer type, and transfer direction of an endpoint. For transfer direction, an endpoint is either a source (IN) or sink (OUT) of the USB device.

String Descriptor: Some of the con□guration descriptors mentioned above can include a stringdescriptorindexnumber.ThehostPCcanthenrequestthe unicode encodedstring for a speci□ed index. This provides the host with human readable information about the device, including stringsfor manufacturer name, product name, and serialnumber.

Device Class

USBdevicesvarygreatlyin termsof function and communication requirements. Somedevices are single–purpose, such asa mouse orkeyboard. Otherdevices mayhavemultiple functional–ities that are accessible via USB such as a printer/scanner/fax device.

The USB-IFDeviceWorking Group de□nesa discreetnumberofdevice classes. The ideawas to simplify softwaredevelopmentbyspecifyinga minimum setof functionality and characteris-ticsthatis sharedby agroupofdevicesandinterfaces.Devicesofthe sameclasscanalluse the same USB driver. Thisgreatly simpli □ es the use of USB devices and saves the end-user thetimeand hassleof installingadriverforeverysingleUSBdevicethatis connectedtotheir hostPC.

Forexample,inputdevicessuchasmice,keyboardsandjoysticks areallpartoftheHID (Human Interface Device) class. Another example is the Mass Storage class which covers removable hard drives andkeychain □ash disks. All of these devices use the same Mass Storage driver which simpli□es their use.

However, a device does not necessarily need to belong to a speci□c device class. In these cases, the USB device will require its own USB driver that the host PC must load to make the functionality available to the host.

On-The-Go(OTG)

TheOTG supplementtotheUSB2.0specprovides methodsfor mobiledevicesto communicate witheachother,activelyswitchtheroleofhostanddevice,andalsorequest sessionsfromeach other when power to the USB is removed.

The initial role of host and device is determined entirelybythe USB connector itself. AllOTG capable peripherals will have a 5-pin Micro-AB receptacle which can receive either the Micro-A or Micro-B plug. If the peripheral receives the Micro-A plug, then it behaves as the host. If it receives the Micro-B plug, then it behaves asthe device. However, there may be certain situations where a peripheral received the Micro-B plug, but needs to behave as the host. Rather than request that the user swap the cable orientation, the two peripherals have the ability to swap the roles of host and device through the Host Negotiation Protocol (HNP).

The HNP begins when the A-device □nishes using thebus and stops allbus activity. The B-device detects this and will release its pull-up resistor. When the A-device detects the SE0, it respondsbyactivating its pull-up. Once the B-device detects this condition, the B-device issues reset and begins standard USB communication as the host.

In order to conserve power, A-devices are allowed to stop providing power to the USB. However, there couldbe situations wheretheB-devicewantstousethebusandV_{BUS} isturnedoff.It is for this reason that the OTG supplement describes a method for allowing the B-device to request a session from the A-device. Upon successful completion of the Session Request Protocol (SRP),the A-device willpowerthebusand continue standardUSBtransactions.

The SRP is broken up into two stages. From a disconnected state, the B-device must begin an SRP by driving one of its data lines high for a suf□cient duration. This is called data-line pulsing. If the A-device does not respond to this, the B-device will drive the V_{BUS} above a speci□ed threshold and release it, therebycompletingV_{BUS} pulsing. If the A-device still does

not begin a session, the B-device may start the SRP over again, provided the correct initial conditions are met.

For more details on OTG, please see the *On-The-Go Supplement to the USB 2.0 Specification*.

References

. USB Implementers' Forum

TOTAL PHASE

1.2 I²C Background

I²C History

When connecting multiple devices to a microcontroller, the address and data lines of each devices were conventionally connected individually. This would take up precious pins on the microcontroller, result in a lot of traces on the PCB, and require more components to connect everything together. This made these systems expensive to produce and susceptible to inter–ference and noise.

To solve this problem, Philips developed Inter–IC bus, or I²C, in the 1980s. I²C is a low–bandwidth, short distance protocol for on board communications. All devices are connected through two wires: serial data (SDA) and serial clock(SCL).

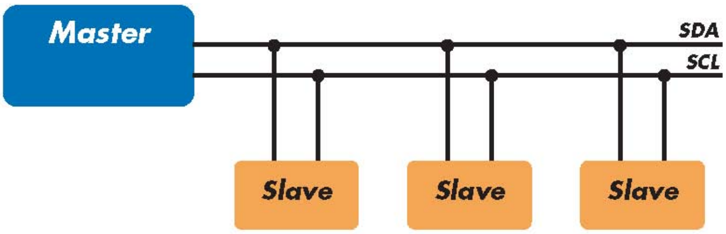

***Figure 12:** Sample I²C Implementation.*
Regardless of how many slave units are attached to the I²C bus, there are only two signals connected to all of them. Consequently, there is additional overhead because an addressing mechanism is required for the master device to communicate with a specific slave device.

Because all commnication takes place on only two wires, all devices must have a unique ad–dressto identifyitonthebus. Slavedeviceshavea prede□ned address,butthelower bitsof the address canbe assignedto allowformultiplesofthe samedevicesonthebus.

I²C Theoryof Operation

I²C has a master/slave protocol. The master initiates the communication. Here is a simpli□ed descriptionofthe protocol.Forprecise details,pleaserefertothePhilips I²C speci□cation.The sequenceofevents are asfollows:

1 The master device issues a start condition. This condition informs all the slave devices to listen on the serial data linefor their respective address.
2 The master device sends the address of the target slave device and a read/write □ag.
3 The slave device with the matching address responds with an acknowledgment signal.
4 Communication proceeds between the master and the slave on the databus. Both the master and slave can receive or transmit data depending on whether the communication isareadorwrite.Thetransmitter sends8bitsofdatatothe receiver, which replieswith a1bit acknowledgment.
5 When the communication is complete, the master issues a stop condition indicating that everything is done.

Figure13 showsa sample bitstreamof the I^2Cprotocol.

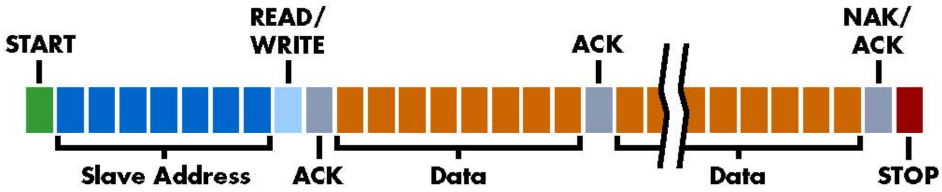

Figure 13: *I^2C Protocol.*
Since there are only two wires, this protocol includes the extra overhead of the addressing and acknowledgement mechanisms.

I^2C Features

I^2C has many features other important features worth mentioning. It supports multiple data speeds: standard (100 kbps),fast (400 kbps) and high speed (3.4 Mbps) communications.

Otherfeatures include:

- Built in collision detection,
- 10-bit Addressing,
- Multi-master support,
- Data broadcast (general call).

For moreinformation about otherfeatures,seethereferencesattheendof this section.

I^2C Benefits and Drawbacks

Sinceonlytwo wires are required, I^2C iswell suitedfor boards with manydevices connected on thebus. This helps reduce the cost and complexity of the circuit as additional devices are added to the system.

Due to the presence of only two wires, there is additional complexity in handling the overhead of addressing and acknowledgments. This can be inefficient in simple configurations and a direct-link interface such as SPI might be preferred.

I^2C References

- I^2C bus – *NXP (Philips) Semiconductors Official I^2C website*
- I^2C (Inter-Integrated Circuit) BusTechnical Overview andFrequently Asked Questions– *Embedded Systems Academy*

Introduction to I^2C – *Embedded.com*

•

I²C – *Open DirectoryProject Listing*

1.3 SPI Background

SPI History

SPI is a serial communication bus developed by Motorola. It is a full–duplex protocol which functions on a master–slave paradigm that is ideally suited to data streaming applications.

SPI Theoryof Operation

SPI requiresfour signals: clock (SCLK), master output/slave input (MOSI), master input/slave output (MISO), slave select (SS).

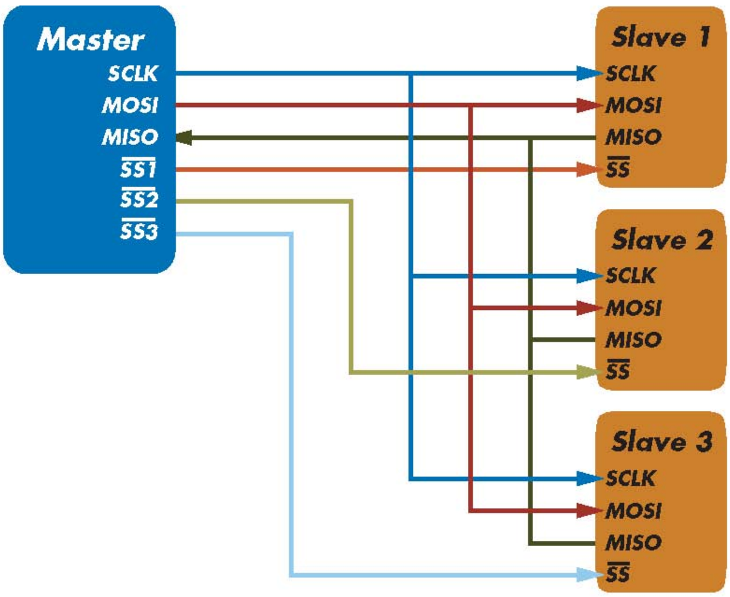

Figure 14: *Sample SPI Implementation.*
Each slave device requires a separate slave select signal (SS). This means that as devices are added, the circuit increases in complexity.

Three signals are shared by all devices on the SPI bus: SCLK, MOSI and MISO. SCLK is generatedbythe master device and is usedfor synchronization. MOSI and MISO are the data lines. The direction of transfer is indicatedbytheir names. Data is always transferred in both directions in SPI,but an SPI device interested in only transmitting data can choose to ignore the receivebytes.Likewise,adeviceonly interestedinthe incomingbytescantransmitdummy bytes.

Each device has its own SS line. The master pulls low on a slave's SS line to select a device for communication.

Theexchange itselfhasno pre-defined protocol. Thismakesit idealfor data-streaming appli-cations. Data can be transferred at high speed, often into the range of the tens of megahertz. The flipside is that there is no acknowledgment, no flow control, and the master maynot even be aware of the slave's presence.

SPI Modes

Although there is no protocol, the master and slave need to agree about the data framefor the exchange. The data frame is describedby two parameters: clock polarity (CPOL) and clock phase (CPHA). Both parameters have two states which results infour possible combinations. These combinations are shown in Figure 15.

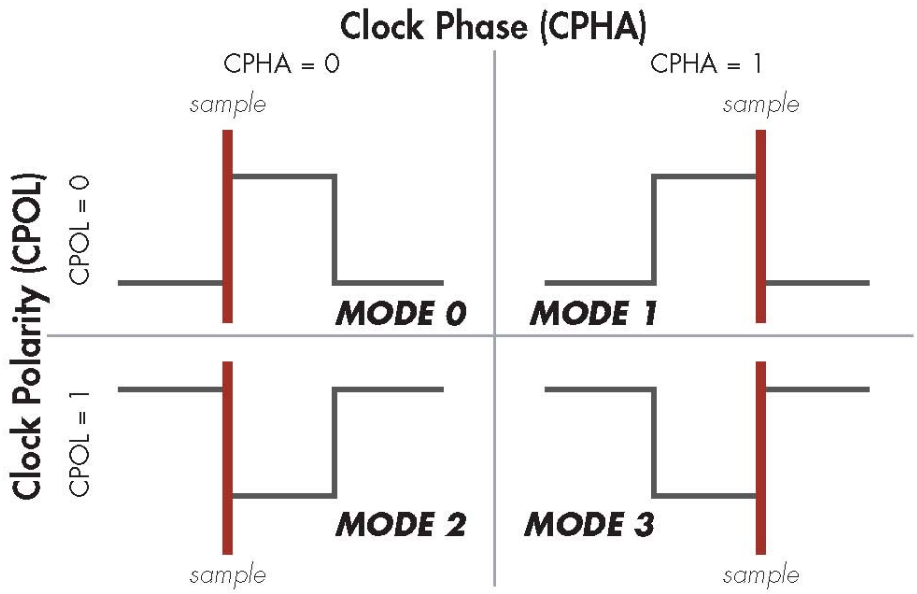

Figure 15: *SPI Modes*
The frame of the data exchange is described by two parameters, the clock polarity (CPOL) and the clock phase (CPHA). This diagram shows the four possible states for these parameters and the corresponding mode in SPI.

SPI Benefits and Drawbacks

SPI is a very simple communication protocol. It does not have a specific high-level protocol which means that there is almost no overhead. Data can be shifted at very high rates in full duplex. This makes it verysimple and efficient in a single master single slave scenario.

Because each slave needs its own SS, the number of traces required is $n+3$, where $n$is the number of SPI devices. This means increased board complexity when the number of slaves is increased.

SPI References

Introduction to SerialPeripheral Interface – *Embedded.com*
SPI – SerialPeripheral Interface

1.4 MDIO Background

MDIO History

Management Data Input/Output,or MDIO,isa 2–wire serialbus thatis usedto manage PHYsor physical layer devices in media access controllers (MACs) in Gigabit Ethernet equipment. The management of these PHYs is based on the access and modi□cation of their various registers.

MDIO was originally de□ned in Clause 22 of IEEE RFC802.3. In the original speci□cation, a single MDIO interface is able to access up to 32 registers in32different PHY devices. These registers provide status and control information such as: link status, speed ability and selec–tion, power downfor low power consumption, duplex mode (full or half), auto–negotiation,fault signaling, and loopback.

To meet the needs the expanding needs of 10–Gigabit Ethernet devices, Clause 45 of the 802.3ae speci□cation provided thefollowing additions to MDIO:

- Ability to access 65,536 registers in 32 different devices on 32 different ports
- Additional OP–code and ST–codefor Indirect Address register accessfor 10 Gigabit Eth–ernet
- End–to–endfault signaling
- Multiple loopbackpoints
- Low voltage electrical speci□cation

MDIO Theoryof Operation

The MDIO bus has two signals: Management Data Clock (MDC) and Managment Data In–put/Ouput (MDIO).

MDIO has speci□c terminology to de□ne the various devices on the bus. The device driving the MDIObusis identi□ed as the Station Management Entity(STA). The targetdevices that are being managedbytheMDC arereferredtoas MDIO ManageableDevices (MMD).

The STAinitiates all communication in MDIO and is responsiblefor driving the clock on MDC. MDC is speci□ed to have a frequency of up to 2.5 MHz.

Clause22

Clause22 de□nes the MDIO communication basicframeformat (Figure 16)whichis composed of thefollowing elements:

Theframeformatonlyallowsa5–bitnumberforboththePHY addressandtheregister address, which limits the number of MMDs that the STAcan interface. Additionally, Clause 22 MDIO only supports 5V tolerant devices and does not have a low voltage option.

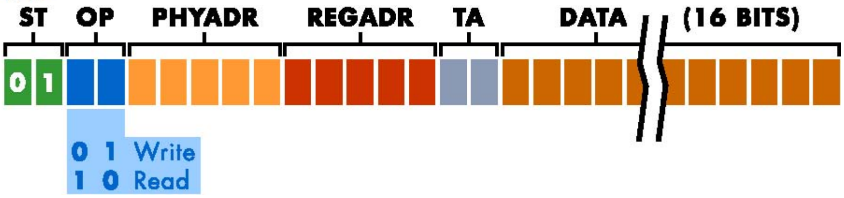

Figure16:Basic MDIOFrameFormatTable3:Clause 22format

ST	2bits	Start of Frame (01 for Clause 22)
OP	2bits	OP Code
PHYADR	5bits	PHY Address
REGADR	5bits	Register Address
TA	2bits	Turnaround time to change bus ownership from STAto MMD if required
DATA	16 bits	Data Driven bySTAduring write Driven byMMD during read

Clause45

In order to address the de□ciencies of Clause 22, Clause 45 was added to the 802.3 speci□ca-tion. Clause45 added supportforlowvoltagedevicesdownto 1.2Vandextendedtheframe format (Figure17)to provide access to manymore devices and registers. Some of the elements of the extended frame are similar to the basic data frame:

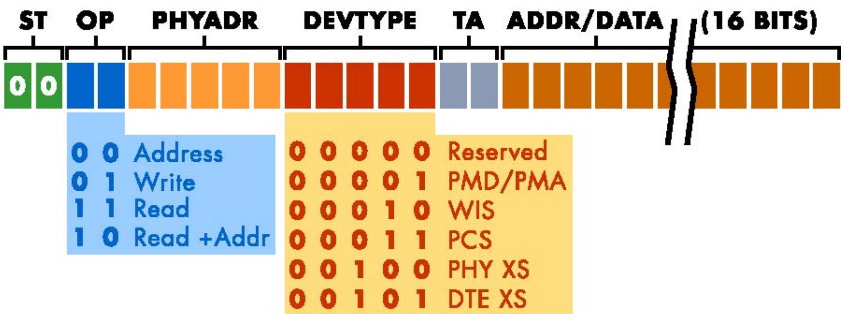

Figure 17: *Extended MDIO Frame Format*

The primary change in Clause 45 is how the registers are accessed. In Clauses 22, a sin-gle frame speci□ed both the address and the data to read or write. Clause 45 changes this paradigm. First an address frame is sent to specify the MMD and register. Asecond frame is then sent to perform the read or write.

The bene□ts of adding this two cycle access are that Clause 45 is backwards compatible with Clause22, allowingdevicesto interoperate with each other. Secondly,bycreatinga address frame, the register address space is increased from5 bits to 16 bits, which allows an STA to

Table4:Clause 45format

ST	2bits	Start of Frame (00 for Clause 45)
OP	2bits	OP Code
PHYADR	5bits	PHY Address
DEVTYPE	5bits	Device Type
TA	2bits	Turnaround time to change bus ownership from STAto MMD if required
ADDR/DATA	16 bits	Address or Data Driven bySTAfor address Driven bySTAduring write Driven byMMD during read Driven byMMD during read-increment-address

access 65,536 different registers.

In orderto accomplishthis, several changesweremadeinthe compositionofthedataframe.A new ST code (00) is de□ned to identify Clause 45 data frames. The OP codes were expanded to specify an addressframe,a writeframe,a readframe, ora read and post read increment address frame. Since the register address is no longer needed, this □eld is replaced with DEVTYPE to specify the targeted device type. The expanded device type allows the STA to access other devices in addition to PHYs.

Additional details about Clause45 canbefound on the IEEE 802.3workgroupwebsite.

MDIO References

IEEE 802 LAN/MAN Standards Committee
Use The MDIO BusTo Interrogate ComplexDevices – *Electronic Design Magazine*

2 Hardware Speci□cations

2.1 Beagle USB 480 Protocol Analyzer

Connector Speci□cation

On one side of the Beagle USB 480 monitor is a single USB-B receptacle. This is the Analysis side (Figure 18). This port connects to the analysis computer that is running the Beagle Data Center software or custom application. Furthermore, the Beagle USB 480 analyzer Analysis side must be plugged in at anytime a target device is plugged in. This is to ensure that all connections are properly powered.

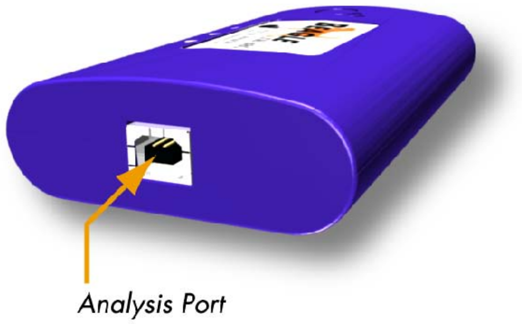

Figure 18: *Beagle USB 480 Protocol Analyzer - Analysis Side*

The opposite side is the Capture side (Figure 19), and it contains a USB–A and USB–B recep–tacle. These are used to connect the target host computer to the target device. The target host computer can be the same computer as the analysis computer, although it maynot be optimal under certain conditions.

Figure 19: *Beagle USB 480 Protocol Analyzer - Capture Side*

The Capture side acts as a USB pass–through. In order to remain within the USB 2.0 speci□–cations, no more than5 meters of USB cable should be used in total between the target host

computer and the target device.

The Capture side also includes a mini–DIN9 connector which serves asa connection to the digital inputs and outputs. Its pin outs are described in Figure 20 and the cable coloringfor the included cable are describedinTable 5.

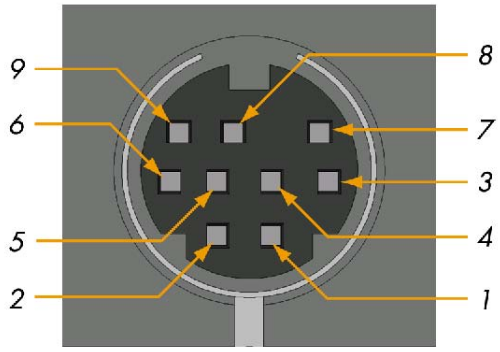

Figure 20: *Beagle USB 480 Protocol Analyzer - Digital I/O Port Pinout*

Table5:Digital I/O Cable Pin Assignments

Pin Name	Color
Input1	Brown
Input2	Red
Input3	Orange
Input4	Yellow
Output1	Green
Output2	

Blue

Output3

Purple

Output4

Grey

Ground

Black

Pin Number

Pin1 Pin2 Pin3 Pin4
Pin5 Pin6 Pin7 Pin8

Pin9

The top of the Beagle USB 480 Protocol Analyzer has three LED indicators as shown in Figure21. Thegreen LED serves as an AnalysisPort connection indicator. Thegreen LED willbe illuminated when the Beagle analyzer has been correctly connected to the analysis computer andis receivingpowerfromUSB.TheamberLEDservesasaTargetHostconnection indicator. The amber LED will be illuminated when the target host computer is connected to the analyzer. Finally,theredLEDisan activityLED.Itsblinkrateis proportionaltothe amountofdatabeing sent across the monitored bus. If no data is seen on the bus, but the capture is active, the activity LED will simply remain on.

Pleasecheckallthe connectionsifthegreenortheamberLEDfailto illuminateaftertheBeagle USB 480 analyzer has been connected to the analysis computer and the target host computer.

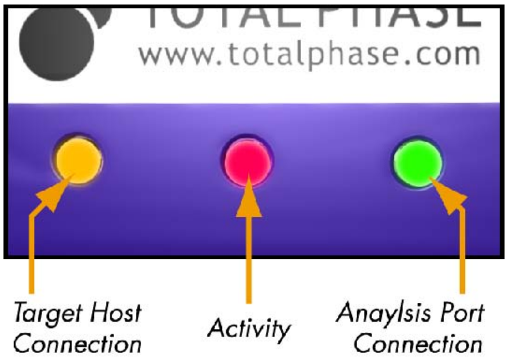

Figure 21: *Beagle USB 480 Protocol Analyzer - LED Indicators*

Digital I/O

Digital inputs allow users to synchronize external logic with the analyzed USB data stream. Whenever the state of an enabled digital input changes, an event will be sent to the analysis PC.The digitalinputmaynot oscillateataratefasterthan30MHz.Anyfasterandtheevents may not be passed to the PC. Also, when an active data packet is on thebus, only one input event will be recorded and sent back to the analysis PC. Once the packet has completed, the latest stateofthelines(if changed)willbesentbacktothePC.Digitalinputsareratedfor3.3V.

Digital outputs allow users to output events to external devices, such as an oscilloscope or logic analyzer, especially to trigger the oscilloscope to capture data. Digital outputs can be set to activate on various conditions that are described more thoroughly in Section 3.3. The digital outputs areratedto3.3Vand10mA.

On-boardBuffer

TheBeagleUSB480analyzer containsa64MB on-boardbuffer. Thisbufferservestwopur-poses. It helps buffer large data □ ows during real-time capture when the analysis computer cannotstreamthedataofftheBeagleanalyzerfast enough.Itisalsouseddduringadelayed-download capture to store all of the captured data.

Hardware Filters

The Beagle USB 480 analyzer provides six different hardware □lters. These will □lter out data-less transactions in the hardware, such as IN+NAKand PING+NAKcombinations. The unwanted dataisthrownaway, reducingthe amountof captureddataonthedevice,the amountof analysis traf□ cbacktothe analysisPC,andthe processingoverheadonthe analysisPC.Amore detailed overview of the hardware □lters is available in Section 3.3.

Signal Speci□cations/Power Consumption

Speed

The Beagle USB 480 Protocol Analyzer supports capture of all wired USB speeds. The analyzer has automatic speed detection as well as manual speed locking.

ESDProtection

The Beagle analyzer has built-in electrostatic discharge protection to prevent damage to the unit from high voltage static electricity.

Power consumption

When the Beagle analyzer is connected, it consumes a maximum of approximately 2.5 mA from the capture host. This is a minimal overhead in addition to the current draw of the target device. Note that if a capture target reports itself as a 100 mA device and draws almost all of that current, the Beagle analyzer's extra power consumption may cause the overall power consumption to be out of spec.

The Beagle analyzer consumes a maximum of approximately 180 mA.

2.2 Beagle USB 12 Protocol Analyzer

Connector Speci□cation

On one side of the Beagle USB 12 monitor is a single USB-B receptacle. This is the Analysis side (Figure 22). This port connects to the analysis computer that is running the Beagle Data Center software.

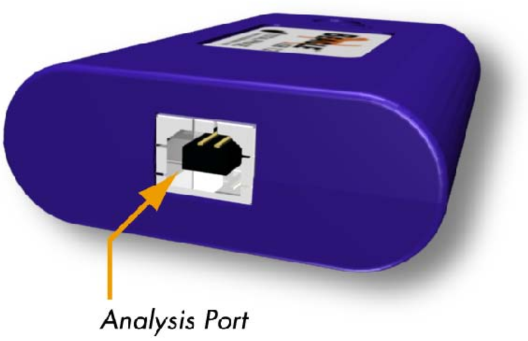

Figure 22: *Beagle USB 12 Protocol Analyzer - Analysis Side*

On the opposite side is the Capture side (Figure 23), are a USB-A and USB-B receptacle. These are used to connect the target host computer to the target device. The target host computer can be the same computer as the analysis computer.

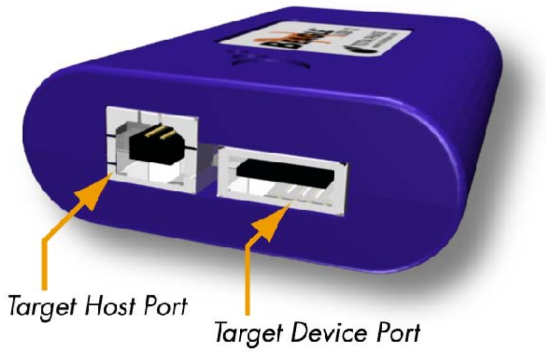

Figure 23: Beagle USB 12 Protocol Analyzer - Capture Side

The Capture side acts as a USB pass-through. In order to remain within the USB 2.0 speci□-cations, no more than5 meters of USB cable should be used in total between the target host computer and the target device. The Beagle USB 12 monitor is galvanically isolated from the USBbus to ensure the signal integrity.

Please note, that on the Capture side, there is a small gap between the two receptacles. In thisgap,twoLED indicators are visible, agreen oneandan amber one, asshownin Figure 24. When the Beagle USB 12 monitor has been correctly connected to the analysis computer, the green LED will illuminate. When the Beagle USB 12 monitor is correctly connected to the target host computer, the amber LED will illuminate.

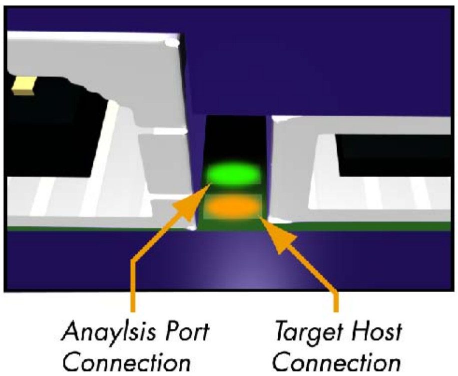

Figure 24: Beagle USB 12 Protocol Analyzer - LED Indicators

Pleasecheckallthe connectionsiftheoneorbothLEDsfailto illuminateaftertheBeagleUSB 12 monitor has been connected to the analysis computer or the target host computer.

Speed

The Beagle USB 12 Protocol Analyzer supports full-and low-speed capture. It does not support high-speed protocolsfor capture.Theuplinktothe analysisPCmustbe high-speed.

ESDprotection

The Beagle analyzer has built-in electrostatic discharge protection to prevent damage to the unit from high voltage static electricity.

Power consumption

The Beagle analyzer consumes a maximum of approximately 15 mA from the capture host. This is a minimal overhead in addition to the current draw of the target device. Note that if a capture target reports itself as a 100 mA device and draws almost all of that current, the Beagle analyzer'sextrapower consumptionwill causetheoverallpower consumptiontobeoutofspec.

Furthermore, the Beagle analyzer consumes a maximum of approximately 125 mA of power from the analysis PC. However, it reports itself to the analysis PC as a low-power device. This reportingallowstheBeagleanalyzertobeusedwhenits analysisportis connectedtoabus-powered hub (which are only technically speci□ed to supply 100 mA per port). Normally this extra amount of power consumption should not cause any serious problems since other ports onthehubaremostlikelynotusingtheirfull100mAbudget.Iftherearanyconcerns regarding the total amount of available current supply, it is advisable to plug the Beagle analyzer's directly into the analysis PC's USB host port or to use a self-powered hub.

2.3 BeagleI[2]C/SPI/MDIOProtocol Analyzer

Connector Speci□cation

The ribbon cable connector is a standard 0.100" (2.54mm) pitch IDC type connector. This connector will mate witha standardkeyedboxed header.

Alternatively, split cables are available which connects to the ribbon cable and provides individ-ual leadsfor each pin with or withoutgrabber clips.

Orientation

The ribbon cable pin order follows the standard convention. The red line indicates the □rst position. When lookingatyour Beagle analyzerintheupright position (□gure 25),pin1isinthe top left corner and pin 10 is in the bottom right corner.

Ifyou□ipyourBeagleanalyzerover(□gure 26)suchthatthetextontheserialnumber labelis inthe properupright position,thepin orderisasshowninthefollowingdiagram.

Order of Leads

1. SCL

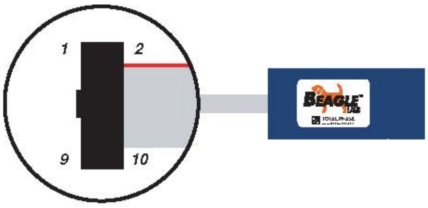

Figure 25: The Beagle I^{2}C/SPI/MDIO Protocol Analyzer in the upright position.
Pin 1 is located in the upper left corner of the connector and Pin 10 is located in the lower right corner of the connector.

1	GND
2	SDA
3	NC/+5V
4	MISO
5	NC/+5V
6	SCLK/MDC
7	MOSI/MDIO
8	SS
9	GND

Ground

GND (Pin 2):GND (Pin 10):

It is imperative that the Beagle analyzer'sground lead is connected to theground of the target system. Withouta common ground betweenthetwo,the signaling willbe unpredictableand communication willlikelybe corrupted.Twogroundpins areprovidedto ensureasecureground path.

I^{2}CPins

SCL (Pin 1):

Serial Clockline – the signal used to synchronize communication between the master and the slave.

SDA(Pin 3):

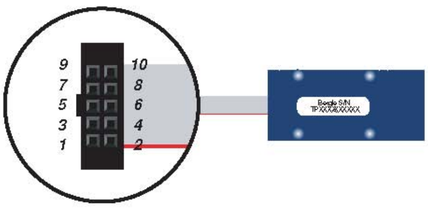

Figure 26: The Beagle I^{2}C/SPI/MDIO Protocol Analyzer in the upside down position.
Pin 1 is located in the lower left corner of the connector and Pin 10 is located in the upper right corner of the connector.

Serial Data line – the bidirectional signal used to transfer data between the transmitter and the receiver.

SPI Pins

SCLK (Pin 7):

Serial Clock – control line that is driven by the master and regulates the flow of the data bits.

MOSI (Pin 8):

Master Out Slave In – this data line supplies output data from the master which is shifted into the slave.

MISO (Pin 5):

Master In Slave Out – this data line supplies the output data from the slave to the input of the master.

SS (Pin 9):

Slave Select – control line that allows slaves to be turned on and off via hardware control.

MDIO Pins

MDC (Pin 7):

Management Data Clock – control line that is driven by the STA and synchronizes the flow of the data on the MDIO line.

MDIO (Pin 8):

Management Data Input/Output – the bidirectional signal used to transfer data between the STA and the MMD.

Powering Downstream Devices

It is possible to power a downstream target, such as an I^2C or SPI EEPROM with the Beagle analyzer's power (which is provided by the analysis PC's USB port). It is ideal if the downstream device does not consume more than 20–30 mA. The Beagle analyzer is compatible with USB hubs as well as USB host controllers. Bus-powered USB hubs are technically only rated to provide 100 mA per USB device. If the Beagle analyzer is directly plugged into a USB host controller or a self-powered USB hub, it can theoretically draw up to 500 mA total, leaving approximately 375 mA for any downstream target. However, the Beagle analyzer always reports itself to the host as a low-power device. Therefore, drawing large amounts of current from the host is not advisable.

Signal Specifications/Power Consumption

Speed

The Beagle I^2C/SPI/MDIO is capable of monitoring I^2C bus bit rates of up to 4 MHz, SPI bit rates of up to 24 MHz, and MDIO bit rates of up to 2.5 MHz. Both I^2C and MDIO monitoring can sustain their respective maximum speeds, however SPI monitoring at the maximum bit rate maynotbepossiblefor sustainedtraf c.Theexact limitationsofSPI monitoringare dependent on the targetbus conditions and the CPUof the hostPC.Forexample, theworst-case situation isa sustained sequenceof shortSPIpacketsatthe maximumbusbitrateof24 MHz.

Itisimportanttonotethatinordertoproperly captureI²C,SPI,orMDIOsignals,the sampling ratemustbeset properly.ForSPIorMDIO monitoring,the minimum requirementforthe sam-plingrateis twice thebus bitrate.ForI²Cmonitoring, the samplingrate shouldbe 5–10 times thebusbitrate.Forfurther detailson thisreferto Section 3.3.

Logic High Levels

All signallevels shouldbe nominally 3.3V(+/−10%) logic high. This allows the Beagle analyzer tobe used with bothTTL(5V)and CMOSlogiclevel(3.3V)devices.Alogichighof3.3Vwill be adequate for TTL-compliant devices since such devices are ordinarily speci ed to accept logic high inputs above approximately3V.

ESDprotection

The Beagle analyzer has built-in electrostatic discharge protection to prevent damage to the unit from high voltage static electricity. This adds a small amount of parasitic capacitance (approximately 15 pF) to the signal path under analysis.

Power Consumption

The Beagle analyzer consumes approximately 125 mA of power from the analysis PC. However, it reports itself to the analysis PC as a low-power device. This reporting allows the Beagle analyzer to be used when its analysis port is connected to a bus-powered hub (which are only technically speci ed to supply 100 mA per port). Normally this extra amount of power consumption should not cause any serious problems since other ports on the hub are most likely not using their full 100 mAbudget. If there are any concerns regarding the total amount of available current supply, it is advisable to plug the Beagle analyzer's directly into the analysis PC's USB host port or to use a self-powered hub.

2.4 USB 2.0

All Beagle analyzers are high-speed USB 2.0 devices. Theyrequire a high-speed USB 2.0 host controllerfor the analysis data connection.

2.5 Temperature Speci cations

The Beagle analyzers are designed to be operated at room temperature (10–35°C). The elec-tronic components are rated for standard commercial speci cations (0–70°C). However, the plastic housing, along with the ribbon and USB cables, maynot withstand the higher end of this range. Any use of the Beagle analyzer outside the room temperature speci cation will void the hardwarewarranty.

3 Device Operation

3.1 Electrical Connections

Beagle USB Protocol Analyzers

The Beagle USB analyzer's analysis port must be connected to the analysis computer through a USB cable. The Capture side of the Beagle analyzer must be placed on the USB to be monitored. Normally, this is accomplishedbyplacing the Beagle analyzer in-line between the USBdeviceandhostbeing monitored. In otherwords,thebustobe monitoredgoes through the Beagle USB analyzer. To properly accomplish this connection, connect the target host to the USB-B receptacle on the Capture side of the Beagle USB analyzer, and connect the target device to the USB-A receptacle on the Capture side of the Beagle USB analyzer. See Section 2.1for more details. Thisis the setup illustratedin panels a–cof Figure27.

Insome cases,thetargetbusisfullyinternaltoan embeddedsystem.Ifso,itissimply neces-saryto tap off the lines through the use of a parallel connector. One can plug in the tapped off cable into either theTarget host orTargetdevice portof the analyzer;both are equivalent. This is illustrated in Figure 27d.

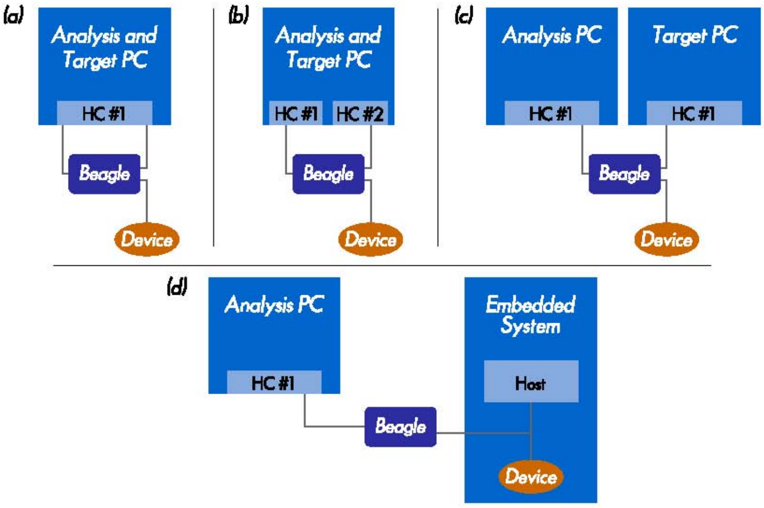

Figure 27: *Beagle USB Protocol Analyzer Connections*
Beagle USB analyzer may be connected to the same bus as it is monitoring (panel a), or to a different bus (panel b). Multiple host controllers may reside in separate host controllers (panel c). Panel d shows the case of sniffing a self-contained embedded bus.

The connections of the Beagle USB analyzer are complicated somewhatby thefact that the Beagle analyzer is monitoring USB signals and then communicating the monitored data back though another USB port. Thus, the issue of the host broadcasting, as described in Section 1.1, comes intoplay. Whileall Beagle analyzers use high-speedUSB communicationrate, this issue is only pertinent when using the Beagle USB 480 Protocol Analyzer to monitora high-speed device. If the Beagle USB 480 Protocol Analyzer's analysis port is connected to the same host controller as a high-speed device that it is monitoring (Figure 27a) then the Beagle analyzer will endup snif□ng someofitsowntraf□c. Thisis especiallytrueifthe Beagle analyzeris con□gured to stream back bus traf□c to the PC in real time! This will be seen in the capture as many INpackets to the Beagle analyzer's device address with occasional downstream handshake packets.

This phenomenon has two negative consequences. Theextra traf□c on the capturebus from the Beagle USB 480 analyzer maymake it dif□cultto locate the USB traf□c of interest within the volume of data captured. Additionally, thebus traf□cfor Beagle USB 480 analyzer will reduce the bandwidthavailableto otherUSBdevicesonthebus.

There are a number of ways to deal with this issue.

One methodfordealingwiththisproblemis install anotherUSBhost controllertothe computer and connect one host controller to the analysis port of the Beagle analyzer and use the other host controller to communicate with the host and device under test (Figure 27b). Downstream USB packets are only broadcast on USB links on the same host controller, so this technique is another way to ensure that the Beagle analyzer's traf□c is not seen on the capture side of the analyzer. The disadvantageis that thePCmust spend processing timefor communicating both with the target device as well as the Beagle analyzer.

The preferred method is to connect separate computers to the analysis port and to the target host port of the Beagle USB 480 analyzer (Figure 27c). This puts the analysis end of the Beagleanalyzeronadifferentbus, ensuringthatitstraf□cisnot seenonthe capturesideofthe analyzer. Furthermore, the analysis PC can have full resources to process the incoming data, and the testPC will notbe encumberedbythe analysis software.

Note: All of the USB ports on most computers are on a single host controller, so connecting to a different USB port is not suf□cient. Installing a PCI, PCI Express, or PC Card USB controller card will ensure there is a second USB host controller on the computer.

If the user is constrained to the scenario illustrated in Figure 27a, there are twofeatures of the Beagle analyzer to help mitigate the dilemmas previously outlined. One is a hardware □ltering optionthatrunsontheBeagleanalyzerto □ lterpackets directedtotheBeagleanalyzer'sdevice address. These packets will be □ltered out from the capturebythe hardware, so it will not be sent backthrough the analysis port. This option does not entirely remove the Beagle analyzer's traf□c from the monitored bus, but it will de□nitely minimize the analyzer's effect on the bus since the INand ACKtokens sent to the analyzer will not again appear in the analysis traf□c. In situations where the maximum bandwidthis requiredby the targetdevice,avoid using this option.The secondfeatureisthe abilitytoperformadelayed–downloadcapture.Inthis capture mode, the capture data is not streamed out of the analysis port of the Beagle analyzer until after the analyzer has stopped monitoring thebus. Thisgreatly reduces the amount of USB traf□c going to the Beagle USB 480 analyzer while the capture is active. Thesefeatures are mentioned later in this section where appropriate.

BeagleI^{2}C/SPI/MDIOProtocol Analyzer

The BeagleI^{2}C/SPI/MDIO analyzer usesa standardUSBcableto connectthe protocol analyzer to the analysis computer. The data line(s), clock, andground of the communication protocol in questionmustbe properly connectedtotheBeagleanalyzer'sdata line(s),clock,andground, respectively.

TOTAL PHASE

3.2 Software Operational Overview

There area seriesof steps requiredfora successful capture. These steps are handledbythe Beagle Data Center software automatically, but must be explicitly followed by an application programmer wishing to write custom software. The application programmer interface (API) is documentedextensivelyin Section 6,butthefollowingis meanttoprovidea high-leveloverview of the operation of the Beagle analyzers.

1 Determine the port number of the Beagle analyzer. The function `bg_find_devices()` returnsa listofport numbersforall Beagle analyzers that are attachedtothe analysis computer.
2 Obtaina Beagle handlebycalling `bg_open()` on the appropriate port number. All other software operations are based on this Beagle handle.
3 Con□gure the Beagle analyzer as necessary. The API documentation provides complete details about the different con□gurations.
4 Start the capturebycalling the `bg_enable()` function.
5 Retrievemonitoreddatabyusingtheread functionsthatare appropriateforthe monitored bus type. There are different functions available for retrieving additional data such as byte-and bit-level timing.
6 End the capture by calling the `bg_disable()` function. At this point the capture is stopped, and no new data can be obtained.
7 Close the Beagle handle with the `bg_close()` function.

If the Beagle analyzer is disabled and then re-enabled it does not need to be re-con□gured. However, upon closing the handle, all con□guration settings will be lost.

Example codeisavailablefordownload from theTotal Phasewebsite. Theseexamples demon-stratehowtoperformthe steps outlineabovefor eachofthe serialbus protocols supported.

3.3 Beagle USB 480 Protocol Analyzer Speci□cs

Asidefrom standardreal-time capture,theBeagleUSB480analyzerprovidesanumberofother features. These features include bus event monitoring, digital inputs and outputs, hardware □ltering, as well as multiple capture modes.

Bus Events

The Beagle USB 480 analyzer provides users with insight into events that occur on the bus. Thesebusevents include suspend, resume, reset, speed changes (including high-speed hand-shake), and connect/disconnect events. Furthermore, events that are unexpected (i.e., don't conform to the USB spec) are tagged with a speci□c status code to bring that to the attention of the user. The Beagle USB 480 analyzer also has the ability to identify imperfect resets, like aTinyJassociated withthe high-speed handshake.ATinyJ(orK)mayalsobe tagged when notina high-speedhandshake situationif the resetis not fullyat an SE0,butis instead □oating above the high-speed receiver threshold. This allows users to see if the host is driving a reset signal that is close enough toground voltage. Alternatively, if this amount of detail on reset signals is not desired, the auto speed-detection could be disabled, and locked to the speci c speedof interest. For more details onUSBbuseventsreferto Section 1.1andtheUSB2.0 spec.

OTG Events

The Beagle USB 480 analyzer has the ability to detect On-The-Go (OTG) events. These events include the Host Negotiation Protocol (HNP) and each stage of the Session Request Protocol (SRP).For more details on these protocols, see Section 1.1.

A HNP event will be returned upon seeing the correct initial conditions, and then detecting a correctly timed SE0followedbythe full-speedJ.If the new host does not issuea reset within the speci ed time, the HNP event will be returned with an error indication.

There are two stages of the SRP, and a separate event is returned for each of them. Upon detecting a data-line pulse, the Beagle software will return an event corresponding to this con-dition. After detectinga data-line pulse, the software will reportaV_{BUS}pulseifitis seen on the bus. Note that this means that anyV_{BUS}pulse that occurs withouta preceding data-line pulse willnotbereportedsinceitis completelyoutoftheOTG speci cation.IftheSRPis successful, it willbefollowedby a host connectevent.Ifitis unsuccessful, thenit willbefollowedby a host disconnectevent.

Digital Inputs

Digital inputs providea meansfor users to insert events into the data stream. There arefour digital inputs that can be enabled individually. Whenever an enabled input changes state it will issueaneventandbetaggedwitha timestampofwhentheinputwasinterpretedbytheBeagle USB 480 analyzer. Digital inputs can not exceed a rate of 30 MHz. Digital inputs that occur fasterthanthatarenotguaranteedtobeinterpreted correctlybytheBeagleanalyzer.Also,only one digital input event may occur per active packet. All other digital input events can only be handled after the packet has completed. Digital inputs, although guaranteed to have the correct timestamp given the previous conditions, have the possibility of being presented out of order becausethey areprovidedrandomlybythe userandhaveno direct correlationtothebus.

It is important to note that the digital inputs are susceptible to cross-talk if they are not being activelydriven. Asituation like this could occurifa digital input has been enabled,but has not been tied to a signal. Anyother nearbysignal (i.e., other digital inputs or outputs) could cause the input to activate. It is recommended that all undriven digital inputs be disabled or tied to ground.

For hardware speci cations of the digital inputs refer to Section2.1.

Digital Outputs

Digital outputs provide a means for users to output certain events to other devices, such as oscilloscopes.Inthisway, userscan synchronizeeventsonthebuswithothersignalsthey may be measuring.

Digital outputs, like digital inputs, are susceptible to cross-talk if left disabled. It is recommended that users do not attempt to use disabled digital outputs on other devices,as their characteristics are not speci□ed. Either disconnect all connections to disabled digital outputs, or tie those outputs toground.

There arefour digital outputs that are user con□gurable. Each digital output has the option of being enabled, active high, or active low. Furthermore, each output can activate on speci□c conditions described below.

Digital Output1 will be asserted whenever the capture is running.
Digital Output2 willbe asserted whenevera packetis detected on thebus.
Digital Output3 will be asserted when the selected PID, device address, and endpoint match.
Digital Output4 will be asserted when the selected PID, device address, endpoint, and data pattern match.

The digital outputs activate as soon as their triggering event can be fully con□rmed. Thus, Pins1and2will activate as soon as the capture activates or rxactive goes high, respectively. However, Pins3and4 must assurea matchofallof their characteristics. Therefore,only once all possible PIDs, device address, and endpoints of a given packet are checked completely can the output activate. The assertionof matched dataonPin4 mustwait untiltheendofthe data packetto assurea match.Packetsthatareshorterthenwhatis de □ nedbythe userto match willactivatePin4ifallthedatauptothatpoint matched correctly.

Hardware speci□cationsfor the digital outputs are providedin Section 2.1.

Hardware Filtering

Hardware □lters provide users with the ability to suppress data-less transactions, like those described in Section 1.1. When possible, the hardware □lters will discard all packetsthat meet the □ltering criteria. These □lters can save a signi□cant amount of capture memorywhen used, and are highly recommended when capture-memorycapacity is a concern.

Another bene□t of the hardware □lters is that they reduce the amount of traf□c between the analysis computer and the Beagle analyzer. This is especially usefulfor situations where the analysis computer hasa hard timekeepingup with the bandwidth requirementsof the Beagle analyzer.Forexample, the analysis computer mayberunning other applications orit mayhave otherdevices attachedtothe samebus.

There are six different hardware □lters that can be used independently or in conjunction with one another. They must simplybe enabledbythe user. Their functionalityis describedbelow.

SOF Filtering will remove all Start-of-Frame(SOF)tokens from the data stream. Please note that enabling the SOF□lter willforfeit the ability to detect suspend and high-speed disconnects conditions on thebus.
IN Filtering will attempt to remove all IN+ACKand IN+NAKpairs.

PING Filtering will attempt to remove all PING+NAKpairs.
PRE Filtering will remove all PREtokens.

SPLIT Filtering will attempt to remove many of the data-less SPLITtransactions. This

□lter will attempt to discard: -SSPLIT+IN(for isochronous and interrupt transfers)
-SSPLIT+IN+ACK(forbulk and control transfers)
-CSPLIT+OUT+NYET-CSPLIT+SETUP+NYET-CSPLIT+IN+NAK-CSPLIT+IN+NYET

Self Filtering will remove all packets intendedfor devices with the same device address as the Beagle analyzer. Due to the architecture of USB, when the Beagle analyzer is snif□ngthe same high-speedbuson whichitis connected,it will seeitsowntraf□con the Capture side (for more details refer to Section 1.1). This □lter gives the user the opportunity to remove that traf□c out of the reported data stream. This □lter, however, is only effective if the Beagle USB 480 analyzer is infact connected to the samebus as it is analyzing. If the Beagle analyzer is connected to a different host controller, this □lter should be disabled, as there is a probability that another device on the Target bus will match the Beagle analyzer's device address, and data to that device will be lost.

Filters and Digital I/O

There are a couple of issues regarding the hardware □ltering and digital I/O that are worth noting. Digital outputs are computed before any □ltering takes place. This means that if an output is set to activate on a normally □ltered packet, the output will still activate even if the packetisnever seenbythe user.Forexample,if SOF□ltering is enabled, digitaloutputs set to activate upon seeing an SOFPID will still activate when an SOFis on thebus.

Digital inputs can potentially invalidate a □lter. The □lters that are susceptible to this are the IN, PING, and SPLIT□lters. These □lters suppress entire transactions based on the sequence of packets on thebus. If an input trigger occurs at any time during this sequence, the entire transaction is sent to the user. As an example of this, if IN+NAKpair □ltering is enabled and a digital input event occurs at anytime between the start of the INtoken and the veryend of the NAKhandshake, the entire transaction will be reported to the user. However, if no digital input event occurs, the IN+NAKpair will be discarded.

Capture Modes

The Beagle USB 480 Protocol Analyzer provides the user with3different capture modes: real-time capture, real-time capture with over□ow protection, and delayed-download.

Real-time Capture

Real-time capture is the default capture mode. It provides the user with real-time status of the bus being monitored. The real-time capture canbe stoppedbythree methods. The □rst method isbyhaving the user end the capture througha bg_disable() call (or though the Beagle Data

Center software). The second method is if the Beagle analyzer loses power. This is not the recommended methodfor stoppinga capture. Finally, the capture willbe automatically stopped bythe Beagle USB 480 analyzer if the 64 MB hardwarebuffer □lls to capacity. In this situation, the Beagle analyzer will no longer capture new data from the monitoredbus. Instead, calls to bg_usb480_read() will only retrieve whatever data is remaining in thebuffer. The last call of bg_usb480_read() will return a BG_READ_USB_END_OF_CAPTUREindicating that the capture has stopped and that there is no new data. The hardwarebuffer may □ll in conditions where the analysis computeris not reading the data from the Beagle analyzer asfast asitis capturing new data.

Real-time Capture with Over□ow Protection

Real-time Capture with Over□ow Protection is essentially identical to real-time capture except that it allows for more ef□cient use of the hardware buffer when it nears full capacity. When the buffer is near capacity, the Beagle USB 480 analyzer will truncate all incoming packets to 4 bytes. The true length of the packet will still be reported to the user, however only the □rst 4 bytes of the given packet will be returned. If the user is using a custom application, the remainder of the packet□eld will be □lled with 0s. However, all packets captured when in truncation mode will be tagged with the BG_READ_USB_TRUNCATION_MODEstatus code bit. Because packets are truncated to4bytes in length, only DATApackets have the potential of being truncated. All tokens, handshakes, etc. will still be shown in their entirety.

This mode truncates large packets reducing further usage of the hardwarebuffer. This allows the analysis PC a chance to siphon more data off of the Beagle analyzer before the hardware buffer becomes completely full. In other words the analysis port can catch up to the target traf□ c.Ifthebufferusagedropsbelowacertain threshold,theanalyzerwill automatically return to normal operation and cease the truncation of long packets.

Delayed-download Capture

Delayed-download capture does not stream data to the analysis computer in real time, but instead saves all of the data in the 64 MB hardwarebuffer until the user is ready to download it. The size of the capture is clearly limited by the hardware buffer's max capacity, so it is recommended to use the hardware □lters to limit data-less transactions when appropriate.

The delayed-download capability will especially bene□t those users that are analyzing high-speed traf□c, but are only using a single computer with a single host controller for both the analysis computer and the target host computer. As described previously, devices on the same host controller must share the available bandwidth. Also, all high-speed devices on the same host controller will see all downstream traf□c. Therefore using delayed-download will limit the Beagle analyzer's participation on thebus. Infact, if no other functions are called between the enable of the capture and the disable, there will be nearly no traf□c at all between the PC and analyzer. The only traf□c will be at the verystart and end of the capture session.

The delayed-download will stop automatically once the buffer has reached capacity. It may also be stopped at any timeby the userby calling the bg_usb480_readfunction. Polling of the status of the buffer is possible through bg_usb480_hw_buffer_stats(), function call. Polling the Beagle analyzer will createtraf□c on thebus, and thus takeup someof theavailable bandwidth. Faster polling rates will clearly take up more bandwidth, and thus if users wish to

minimize their impact on thebus, theyshould not poll thebuffer at all. Regardless, the polling traf□c itself canbe □ltered from the analysis databyusing the hardware based Self Filter.

3.4 Beagle I²C/SPI/MDIOProtocol Analyzer Speci□cs

Sampling Rate

Unlike the Beagle USB analyzers, the sampling rate of the Beagle I^2C/SPI/MDIO analyzer is configurable. In order to accurately capture data the sampling rate must be properly set. For SPI and MDIO analysis all data lines are registered using the clocklineof thebus. The internal sampling clock is then used to retrieve the data. The sampling rate should be set to at least twicethebitrate,butpreferablyfaster(4–5 times)if possible. Higher samplingrates canhave the added benefit of increasing timing precision.

Duetothe architectureof I^2C, there are specificbusevents thatoccur betweenthe standard bit-times. In order to capture these transitions, the bus must be oversampled independent of the clock line of the bus. A sampling rate of five to ten times the bus bit rate is recom-mended. This should not be a problem, however, since the minimum sampling rate of the Beagle I^2C/SPI/MDIO analyzer is 10 MHz, and I^2Cbuses usually operate at less than1 MHz frequencies.

The one caveat to setting the sampling rate to very high values is that higher sampling rates create moretrafficonthe analysisUSBthat connectsthe analyzertothe hostPC. Thismay or maynot affect performance depending on the analysisPC configuration.

4.1 Compatibility

Linux

The Beagle software is compatible with all standard 32-bit distributions of Linux with integrated USB support.Kernel 2.6 orgreateris required.

Windows

The Beagle software is compatible with 32-bit versions of Windows 2000 SP4 and Windows XP SP2. Currently 16-bit and 64-bit versions of Windows are not supported.

4.2 Linux USB Driver

TheBeagle communicationslayer underLinuxdoesnot requirea speci□ckerneldrivertoop-erate. However, the user must ensure independently that the libusb library is installed on the system since the Beagle libraryis dynamically linked to libusb.

Most modern Linux distributions use the udev subsystem to help manipulate the permissions of various system devices. This is the preferred wayto support access to the Beagle analyzer such that thedeviceis accessiblebyallof the users on the system upondevice plug-in.

For legacy systems, there are two different ways to access the Beagle analyzer, through USB hotplug or by mounting the entire USB □lesystem as world writable. Both require that `/proc/bus/usb`is mounted on the system which is the case on most standard distributions.

UDEV

Supportforudevrequiresasingle con□guration□lethatisavailableonthe softwareCD,andalso listed on theTotal Phasewebsitefordownload. This □leis `99-totalphase.rules`. Please follow thefollowing steps to enable the appropriate permissionsfor the Beagle analyzer.

1 As superuser, unpack `99-totalphase.rules`to `/etc/udev/rules.d`
2 `chmod 644 /etc/udev/rules.d/99-totalphase.rules`

3. Unplug and replugyour Beagle analyzer(s)

USB Hotplug

USB hotplug requires two con□guration □les which are available on the software CD, and also listedontheTotal Phasewebsitefordownload. These □les are: `beagle`and `beagle.usermap`. Pleasefollow thefollowing steps to enable hotplugging.

1 As superuser, unpack `beagle`and `beagle.usermap`to `/etc/hotplug/usb`
2 `chmod 755 /etc/hotplug/usb/beagle`

3. `chmod 644 /etc/hotplug/usb/beagle.usermap`

1 Unplug and replugyour Beagle analyzer(s)
2 Set the environment variable USB_DEVFS_PATHto /proc/bus/usb

World-Writable USB Filesystem

Finally, here is a last-ditch method for con□guring your Linux system in the event that your distributiondoesnothaveudevorhotplug capabilities.Thefollowing procedureisnot necessary ifyouwereabletoexercisethe stepsintheprevious subsections.

Often, the /proc/bus/usbdirectoryis mounted with read-write permissionsfor root and read-only permissionsfor all other users. If an non-privileged user wishes to use the Beagle analyzer and software, one must ensure that /proc/bus/usbis mounted with read-write permissions for all users. Thefollowing steps can help setup the correct permissions. Please note that these steps will make the entire USB □lesystem world writable.

1. Checkthe current permissionsby executing thefollowing command: “ls-al/proc/bus/usb/001”

2.Ifthe contentsofthat directoryareonlywritablebyroot,proceedwiththe remainingsteps outlined below.

3. Add thefollowing line to the /etc/fstab□le:

```
none /proc/bus/usb usbfs defaults,devmode=0666 0 0
```

1 Unmount the /proc/bus/usbdirectoryusing “umount”
2 Remount the /proc/bus/usbdirectoryusing “mount”
3 Repeatstep1.Nowthe contentsofthat directoryshouldbewritablebyall users.
4 Set the environment variable USB_DEVFS_PATHto /proc/bus/usb

4.3 Windows USB Driver

The current versionof the Beagle analyzer Windowsdriveris 1.1.0.0. Ifyou receive an error message referring to an incompatible driver, refer to Section 4.3for instructions on uninstalling the Beagle analyzer driver. Then download and install the latest driver from our website.

Driver Installation

On the Windows platform, the Beagle software uses a version of the libusb-win32 open source driver to access the Beagle analyzer. For more information on this driver, please refer to the README.txt that is included with the driver. To install the appropriate USB communication driver under Windows, step through thefollowing instructions. This is only necessaryfor the very□rst Beagle analyzer that is plugged into the PC. Subsequent plugs and unplugs should be automatically handledbythe operating system.

Pleasenote,youmay seeawarningwindowthat statesthatthedriverfortheBeagleanalyzer hasnotpassedWindowsLogoTesting.Itissafeto installthedriver, sopleaseselect “Continue Anyway” to continue installing the driver.

Windows 2000:

1 WhenyoupluginthеBeagleanalyzerintoyourPCforthe□rsttime,Windowswill present the “Found New Hardware Wizard.” Select “Next.”
2 On the next dialog window, select “Search for a suitable driver for my device (recom-mended)” and click“Next.”
3 On the third screen, uncheckall settings and check“Specify a location” and click“Next.”
4 Click“Browse...”,navigateto eithertheCD-ROM(\usb-drivers\windowsdirectory), or temporary directory where the driver □les have been unpacked (for downloaded up-dates).
5 Select “beagle.inf” and click“Open”, then click“OK.”
6 Click“Next” on the subsequent screen,followedby“Finish” to complete the installation. This completes the installation of the USB driver.

Windows XP:

1 WhenyoupluginthеBeagleanalyzerintoyourPCforthe□rsttime,Windowswill present the “Found New Hardware Wizard.”
2 Select “Install from a list or speci□c location (Advanced)” and click“Next.”
3 Select “Searchfor best driver in these locations:”, uncheck “Search removable media”, check“Include this location in the search.”
4 Click “Browse...”, expand My Computer and then navigate to either the CD-ROM (\usb-drivers\windowsdirectory), or temporarydirectorywhere the driver □les have been unpacked (for downloaded updates).
5 Click“OK”, then click“Next.”
6 Adialog will inform the user that the USB driver has been installed. Click“Finish.”

Both Windows 2000 and Windows XP:

1 Once the installationis complete, con□rm that the installationwas successfulbychecking that the device appears in the “Device Manager.” To navigate to the “Device Manager” screen select “ControlPanel|System Properties|Hardware|Device Manager.”
2 The Beagle analyzer should appear under the “LibUSB-Win32 Devices” section.

TOTAL PHASE

Driver Removal

Ordinarily, there is usually no harm in leaving the Beagle analyzer’s USB drivers installed in the operating system. However, if it is necessary that the drivers be removed, please follow the steps outlined below.

1 Pluginthe Beagle analyzer whosedriveryou wishto uninstall.
2 Navigate to the “Device Manager” screenbyselecting “ControlPanel|System Properties |Hardware|Device Manager.”
3 Right click on the Beagle analyzer which should appear under the “LibUSB-Win32 De-vices” section.
4 Open the properties dialog.
5 Select the “Driver” tab and choose “Uninstall.”

6 Repeat steps 1–5for each different type (USB,I^{2}C/SPI/MDIO)of Beagledeviceyou wish to uninstall.

7 Now use the file searchingfeature of Windows to search in c:\WINNT\inffor all files containing the text "Beagle."

8 Delete all files with the extension ".inf".

4.4 USBPort Assignment

The Beagle analyzer is assigned a port on a sequential basis. The first analyzer is assigned to port 0, the second is assigned to port 1, and so on. If a Beagle analyzer is subsequently removed from the system, the remaining analyzers shift their port numbers accordingly. Hence with n Beagle analyzers attached, the allocated ports will be numbered from 0to n−1.

DetectingPorts

As described in following API documentation chapter, the bg_find_devicesroutine can be used to determine the mapping between the physical Beagle analyzers and their port numbers.

4.5 Beagle DynamicallyLinked Library

DLL Philosophy

The Beagle DLL provides a robust approach to allow present-dayBeagle-enabled applications to interoperate with future versions of the device interface software without recompilation. For example, take the case of a graphical application that is written to monitor I^{2}C, SPI, MDIO, or USB through a Beagle analyzer. At the time the program is built, the Beagle software is released as version 1.2. The Beagle interface software may be improved many months later resulting in increased performance and/or reliability; it is now released as version 1.3. The original application need not be altered or recompiled. The user can simply replace the old Beagle DLL with the newer one. How does this work? The application contains only a stub

which in turn dynamically loads the DLL on the first invocation of any Beagle API function. If the DLL is replaced, the application simply loads the new one, thereby utilizing all of the improvements present in the replaced DLL.

On Linux, the DLL is technically known as a shared object (SO).

DLL Location

Total Phase provides language bindings that can be integrated into any custom application. The default behavior of locating the Beagle DLL is dependent on the operating system platform and specific programming language environment.Forexample,foraC or C++ application, the following rules apply:

Ona Linux system thisis asfollows:

1 First, searchfor the shared object in the application binarypath. Note, that this step re-quires /proc

lesystem support, whichis standardin 2.4.xkernels.Ifthe /proc□lesys-tem is not present, this step is skipped.

2 Next, search in the application's current working directory.

3 Search the paths explicitly speci□ed in LD_LIBRARY_PATH.

4 Finally, check anysystem librarypaths as speci□ed in /etc/ld.so.confand cached in /etc/ld.so.cache.

Ona Windows system, thisis asfollows:

1 The directoryfrom which the application binary was loaded.

2 The application's current directory.

3 32-bit system directory. (Ex: c:\winnt\System32)[Windows NT/2000/XP only]

4 16-bit system directory. (Ex: c:\winnt\Systemor c:\windows\system)

5 The windows directory. (Ex: c:\winntor c:\windows)

6 The directories listedin thePATH environmentvariable.

If the DLL is still notfound, the BG_UNABLE_TO_LOAD_LIBRARYerror will be returnedby the binding function.

DLLVersioning

The Beagle DLL checks to ensure that the □rmware of a given Beagle analyzer is compatible. Each DLL revision is tagged as being compatible with □rmware revisionsgreater than or equal to a certain version number. Likewise, each □rmware version is tagged as being compatible withDLLrevisionsgreaterthanorequaltoa speci□cversionnumber.

Here is an example.

TOTAL PHASE

```
DLL v1.20: compatible with Firmware >= v1.15
Firmware v1.30: compatible with DLL >= v1.20
```

Hence, the DLL is not compatible with any □rmware less than version 1.15 and the □rmware is not compatible with any DLL less than version 1.20. In this example, the version number constraints are satis□ed and the DLL can safely connect to the target □rmware without error. If thereisaversion mismatch,theAPIcallstoopenthedevicewillfail.SeetheAPI documentation for further details.

4.6 Rosetta Language Bindings: API Integration into Custom Applications

Overview

The Beagle Rosetta language bindings make integration of the Beagle API into custom appli-cations simple. Accessing a Beagle analyzer's functionality simply requires function calls to the BeagleAPI.ThisAPIiseasyto understand,muchliketheANSIClibraryfunctions,(e.g.,there is no unnecessaryentanglement with the Windows messaging subsystem like development kits for some other embedded tools).

First, choose the Rosetta bindings appropriatefor the programming language. Different Rosetta bindings are included with the software distribution on the distribution CD. They can also be foundin the softwaredownload packageavailable on theTotalPhasewebsite. Currently the following languages are supported: C/C++, Python, Visual Basic6, Visual Basic .NET, and C#. Next, follow the instructions for each language binding on how to integrate the bindings with your application build setup. As an example, the integration for the C language bindings is describedbelow.(Forinformationonhowtointegratethe bindingsfor other languages,please see theexample code included on the distributionCD and alsoavailablefordownload on the Total Phase website.)

1 Include the `beagle.h` file includedwiththeAPIsoftwarepackageinanyCorC++ source module. The module may now use anyBeagle API call listed in `beagle.h`.
2 Compile and link `beagle.c` withyour application. Ensure that the include pathfor com-pilation also lists the directoryin which `beagle.h` is located if the two files are not placed in the same directory.
3 Place the Beagle DLL, included with the API software package, in the same directory as the application executable or in another directory such that it will be found by the previously described search rules.

Versioning

Since a new Beagle DLL can be made available to an already compiled application, it is essen-tialto ensure the compatibilityof the Rosetta binding usedbythe application(e.g., `beagle.c`) against the DLL loaded by the system. A system similar to the one employed for the DLL-Firmware cross-validationis usedfor the binding and DLL compatibility check.

Here is an example.


```
DLL v1.20: compatible with Binding >= v1.10 Binding v1.15:
compatible with DLL >= v1.15
```

The above situation will pass the appropriate version checks. The compatibility check is per-formed within the binding. If there is a version mismatch, the API function will return an error code, `BG_INCOMPATIBLE_LIBRARY`.

Customizations

While provided language bindings stubs are fully functional, it is possible to modify the code found within this file according to specific requirements imposedbythe application designer.

Forexample,intheCbindingsonecanmodifytheDLL searchandloadingbehaviortoconform to a specific paradigm. See the comments in `beagle.c` for more details.

4.7 Application Notes

Receive Saturation

Once enabled, the Beagle analyzer is constantly monitoring data on the targetbus. Between calls to the Beagle API, these messages must be buffered somewhere in memory. This is accomplished on the analysis computer, courtesy of the operating system. Naturally thebuffer is limitedinsizeand oncethisbufferisfull,datawillbe dropped.Anover□owcan occurwhen the Beagle analyzer receives datafaster than therate thatitis processed — the receive link is ‘saturated.” The system is most susceptible to saturation when monitoring large amounts of traf□cover USB or high-speed SPIbus.

Threading

The Beagle DLL is designed for single-threaded environments so as to allow for maximum cross-platform compatibility. If the application design requires multi-threaded use of the Bea-gle analyzer's functionality, each Beagle API call can be wrapped with a thread-safe locking mechanism before and after invocation.

It is the responsibility of the application programmer to ensure that the Beagle analyzer open and close operations are thread-safe and cannot happen concurrently with any other Beagle analyzer operations. However, once a Beagle analyzer is opened, all operations to that device can be dispatched to a separate thread as long as no other threads access that same Beagle analyzer.

5 Firmware

5.1 Philosophy

The □rmware included with the Beagle analyzer provides for the analysis of the supported protocols.Itis installedatthefactoryduring manufacturing. Somepartsofthe□rmware canbe updated automaticallybythe software. Other piecesofthe□rmware requireadeviceupgrade utility. In those cases, the Beagle software automatically detects □rmware compatibility and will inform the user if an upgrade is required.

5.2 Procedure

Firmware upgrades should be conducted using the procedure speci□ed in the README.txt that accompanies the particular □rmware revision.

TOTAL PHASE

6 API Documentation

6.1 Introduction

The API documentation describes the Beagle RosettaCbindings.

6.2 General DataTypes

Thefollowing de□nitions are providedfor convenience. The Beagle API provides both signed and unsigned data types.

```
typedef unsigned char u08; typedef unsigned
short u16; typedef unsigned int u32; typedef
unsigned long long u64; typedef signed char
s08; typedef signed short s16; typedef signed
int s32; typedef signed long long s64;
```

6.3 Notes on Status Codes

Most of the Beagle API functions can return a status or error code back to the caller. The complete list of status codes is provided at the end of this chapter. All of the error codes are assigned values less than 0, separating these responses from any numerical values returned bycertain API functions.

Each API function can return one of two error codes with respect to the loading of the Bea−gle DLL, `BG_UNABLE_TO_LOAD_LIBRARY`and `BG_INCOMPATIBLE_LIBRARY`. If these status codes are received, refer to the previous sections in this datasheet that discuss the DLL and API integration of the Beagle software. Furthermore, all API calls can potentially return the errors `BG_UNABLE_TO_LOAD_DRIVER`or `BG_INCOMPATIBLE_DRIVER`. If either ofthese errors are seen, please make sure the driver is installed and of the correct version. Where appro−priate, compare the language bindingversions(`BG_HEADER_VERSION`found in`beagle.h`and `BG_CFILE_VERSION`found in`beagle.c`)to verify that there are no mismatches. Next, ensure thatthe Rosetta language binding(e.g., `beagle.c`and `beagle.h`)are from the same release as the Beagle DLL. If all of these versions are synchronized and there are still problems, please contactTotal Phase supportfor assistance.

Note that anyAPI function that accepts a Beagle handle can potentially return the error code `BG_INVALID_HANDLE`if the handle does not correspond to a valid Beagle analyzer that has already been opened. If this error is received, check the application code to ensure that the `bg_open`command returnedavalid handle and that this handlewas not corrupted before being passed to the offending API function.

Finally, any API call that communicates with a Beagle analyzer can also return the error `BG_COMMUNICATION_ERROR`. This means that while the Beagle handle is valid and the com−munication channel is open, there was an error communicating with the device. This is possible if the device was unplugged while being used.

If either the I^2C, SPI, MDIO, or USB subsystems have been disabledbybg_disable, all other API functions that interact with I^2C, SPI, MDIO, and USB will return BG_I2C_NOT_ENABLED, BG_SPI_NOT_ENABLED, BG_MDIO_NOT_ENABLED, or BG_USB_NOT_ENABLED, respectively.

These common status responses are not reiteratedfor each function. Only the error codes that are speci□c to each API function are described below.

Allof the possible error codes, along with theirvalues and status strings, are listedfollowing the API documentation.

Interface

Find Devices (bg_nd_devices)

```
int bg_find_devices (int num_devices,
                     u16 devices);

                         *
```

Get a list of ports to which Beagle devices are attached.

Arguments

num_devices: maximum number of devices to return devices:

arrayinto which the port numbers are returned

ReturnValue

This function returns thenumberofdevicesfound, regardlessof the arraysize.

Specic Error Codes

None.

Details

Each element of the arrayis written with the port number.Devices that are in use are OR'ed with BG_PORT_NOT_FREE(0x8000). Under Linux, such

devices correspond to Beagle analyzers that are currently in use. Under Windows, such devicesare currentlyin use,butitis not knownif thedeviceisa Beagle analyzer.Example:

```
Devices are attachedtoport0,1,2 ports0and2
areavailable, and port1is in-use.
array=>{ 0x0000, 0x8001, 0x0002}
```

If the input arrayis NULL, it is not lled with any values.If there are more devices than the arraysize (as speciedby nelem), only the rst nelemportnumbers will be written into the array.

Find Devices (bg_nd_devices_ext)

```
int bg_find_devices_ext (int num_devices,
                         u16 devices,

                            * int num_ids, u32
                         unique_ids);

                            *
```

Get a list of ports and unique IDs to which Beagle devices are attached.

Arguments

num_devices: maximum number of devices to return devices:

arrayinto which the port numbers are returned num_ids:

maximum number of device IDs to return unique_ids:

arrayinto which the unique IDs are returned

ReturnValue

This function returns thenumberofdevicesfound, regardlessof the arraysizes.

Speci c Error Codes

None.

Details

This function is the same as bg_find_devices () except that is also returns the unique IDs of each Beagle device. The IDs are guaranteed to be non−zero if valid.

The IDs are the unsigned integer representation of the 10−digit serial numbers. Thenumberofdevices and IDs returnedin eachof their respective arraysis determinedbythe minimumof num_devicesand num_ids. However, if either arrayis NULL, the length passed in for the other arrayis used as−is, and theNULLarrayis not populated. If both arrays are NULL, neither arrayis populated,but thenumberofdevicesfoundis still returned.

Open a Beagle analyzer (bg_open)

```
Beagle bg_open (int port_number);
```

Open the Beagle port.

Arguments

port_number: The Beagle analyzer port number. This port number is the the same as the one obtained from the bg_find_devices () function. It is a zero−based number.

ReturnValue

This function returnsa Beagle handle, whichis guaranteedtobegreater thanzeroifvalid.

Speci c Error Codes

BG_UNABLE_TO_OPEN: The speci ed port is not connected to a Beagle analyzer or the port is already in use.

BG_INCOMPATIBLE_DEVICE: Thereisaversion mismatchbetweentheDLLandthe hardware. TheDLLisnotofa suf cient versionfor interoperabilitywiththe hardwareversionorvice versa. See bg_open_ext () in Section 6.4for more information.

Details

This functionis recommendedfor usein simple applications whereextended informationis not required.For more complex applications, the useof bg_open_ext () is recommended.

Open a Beagle analyzer (bg_open_ext)

```
Beagle bg_open_ext (int port_number, BeagleExt *bg_ext);
```

Open the Beagle port, returning extended information in the supplied structure.

Arguments

port_number: same as bg_openbg_ext: pointerto

pre-allocatedstructureforextendedversioninformationavailableonopen

ReturnValue

This function returnsa Beagle handle, whichis guaranteedtobegreater thanzeroifvalid.

Speci c Error Codes

BG_UNABLE_TO_OPEN: The speci ed port is not connected to a Beagle analyzer or the port is already in use.

BG_INCOMPATIBLE_DEVICE: Thereisaversion mismatchbetweentheDLLandthe hardware. TheDLLisnotofa suf cient versionfor interoperabilitywiththe hardwareversionorvice versa. Theversioninformationwillbeavailableinthe memorypointedtoby bg_ext.

Details

If 0 is passed as the pointer to the structure bg_ext, this function will behave exactly like bg_open().

The BeagleExtstructure is described below:

```
struct BeagleExt {
     BeagleVersion version;
     /* Feature bitmap for this device. */
     int features;

};
```

The features eld denotes the capabilities of the Beagle analyzer. See the API function bg_featuresfor more information.

The BeagleVersionstructure describes the various version dependencies of Beagle compo-nents. It can be used to determine which component caused an incompatibility error.

```
struct BeagleVersion {
     /* Software, firmware, and hardware versions. */
     u16 software;
     u16 firmware;
     u16 hardware;


    /*
 *Hardware revisions that are compatible with this software version.
 *The top 16 bits gives the maximum accepted hw revision.
 *The lower 16 bits gives the minimum accepted hw revision.

     */
     u32 hw_revs_for_sw;
```

```
/*
* Firmware revisions that are compatible with this software version.
* The top 16 bits gives the maximum accepted fw revision.
* The lower 16 bits gives the minimum accepted fw revision.
*/
u32 fw_revs_for_sw

/*
* Driver revisions that are compatible with this software version.
* The top 16 bits gives the maximum accepted driver revision.
* The lower 16 bits gives the minimum accepted driver revision.
* This version checking is currently only pertinent for WIN32
* platforms.
*/
```



```
u32 drv_revs_for_sw;

/* Software requires that the API must be >= this version. */ u16 api_req_by_sw;
};
```

Allversionnumbers areof theformat:

```
(major « 8) | minor
```

example: v1.20would be encoded as 0x0114.

The structure is zeroed before the open is attempted. It is □lled with whatever information is available. For example, if the hardware version is not □lled, then the device could not be queriedfor itsversionnumber.

This functionis recommendedfor usein complex applications whereextended informationis required.For simpler applications, the useof bg_open() is recommended.

Close a Beagle analyzer connection (bg_close)

```
int bg_close (Beagle beagle);
```

Close the Beagle analyzer port.

Arguments

beagle: handle of a Beagle analyzer to be closed

ReturnValue

The number of analyzers closed is returned on success. This will usually be 1.

Speci□c Error Codes

None.

Details

If the handleargument is zero, the function will attempt to close all possible handles, thereby closing all open Beagle analyzer. The totalnumberof Beagle analyzers closedis returnedby the function.

Get Features (bg_features)

```
int bg_features (Beagle beagle);
```

Return thedevicefeatures asa bit−maskofvalues, or an error codeif the handleis notvalid.

Arguments

`beagle`: handle of a Beagle analyzer

ReturnValue

Thefeaturesof the Beagle analyzer are returned. These area bit−maskof thefollowingvalues.

```
#define BG_FEATURE_NONE (0)
#define BG_FEATURE_I2C (1<<0)
#define BG_FEATURE_SPI (1<<1)
#define BG_FEATURE_USB (1<<2)
#define BG_FEATURE_MDIO (1<<3)
```


Speci c Error Codes

None.

Details

None.

Get FeaturesbyUniqueID (bg_unique_id_to_features)

`intbg_unique_id_to_features(u32unique_id);` *Returnthe bitmaskofdevicefeaturesforthegiven Beagledevice, identi edby* `unique_id`.

Arguments

`beagle`: unique ID of a Beagle analyzer

ReturnValue

Thefeatures of the Beagle analyzer are returned. See `bg_features()` for details on the bit map.

Speci c Error Codes

None.

Details

None.

GetPort (bg_port)

```
int bg_port (Beagle beagle);
```

Return the port numberfor this Beagle handle.

Arguments

`beagle`: handle of a Beagle analyzer

ReturnValue

The port number corresponding to the given handle is returned. It is a zero−based number.

Speci□c Error Codes

None.

Details

None.

Get Unique ID (bg_unique_id)

```
u32 bg_unique_id (Beagle beagle);
```

Return the unique ID of the given Beagle analyzer.

Arguments

`beagle`: handle of a Beagle analyzer

ReturnValue

This function returns the unique ID for this Beagle analyzer. The IDs are guaranteed to be non−zero if valid. The ID is the unsigned integer representation of the 10−digit serial number.

Speci□c Error Codes

None.

Details

None.

Status String (bg_status_string)

```
const char *bg_status_string (int status);
```

Return the status stringfor the given status code.

Arguments

`status`: status code returnedby a Beagle API function

ReturnValue

This function returns a human readable string that corresponds to status. If the code is not valid, it returns a NULL string.

Speci□c Error Codes

None.

Details

None.

Version (bg_version)

```
int bg_version (Beagle beagle, BeagleVersion *version);
```

Returntheversion matrixforthedevice attachedtothegiven handle.

Arguments

beagle: handle of a Beagle analyzerversion: pointer to pre–allocated structure

ReturnValue

ABeagle status code is returned withBG_OKon success.

Speci c Error Codes

BG_COMMUNICATION_ERROR: The rmware of the speci ed device can not be determined.

Details

If the handleis0 orinvalid, only the softwareversionis set.See the details of bg_open_ext () for the de

nition ofBeagleVersion.

TOTAL PHASE

Capture Latency(bg_latency)

```
int bg_latency (Beagle beagle, u32 milliseconds);
```

Set the capture latency to the speci ed number of milliseconds.

Arguments

beagle: handle of a Beagle analyzermilliseconds: new capture latency in milliseconds

ReturnValue

ABeagle status code is returned withBG_OKon success.

Speci c Error Codes

BG_STILL_ACTIVE: An attempt was made to change the con guration while the capture was still active.

Details

Set the capture latency to the speci ed number of milliseconds.

The capture latency effectively splits up the total amount of buffering (as determined by bg_host_buffer_size ())into smaller individual buffers. Only once one of these individ–ualbuffers is lled, does the read function return. Therefore, in order to ful ll shorter latency requirementsthese individualbuffersaresettoa smallersize.Ifalargerlatencyis requested, thenthe individualbufferswillbesettoalargersize.

Settingasmalllatencycan increasethe responsivenessofthereadfunctions.Itisimportantto keepinmindthatthereisa xedcostto processingeachindividualbufferthatis independent ofbuffersize.

Therefore,thetrade–offisthatusinga small latencywill increasetheoverhead *perbyte* buffered.Alarge latency setting decreases thatoverhead,but increases the amountof timethatthelibrary mustwaitforeachbufferto□llbeforethelibrary can processtheir contents.

This setting is distinctly different than the timeout setting. The latency time should be set to a value shorter than the timeout time.

TimeoutValue (bg_timeout)

```
int bg_timeout (Beagle beagle, u32 milliseconds);
```

Set the read timeout to the speci□ed number of milliseconds.

Arguments

`beagle`: handle of a Beagle analyzer`milliseconds`: new timeout value in milliseconds

ReturnValue

ABeagle status code is returned with`BG_OK`on success.

Speci□c Error Codes

None.

Details

Set the idle timeout to the speci□ed number of milliseconds.This function sets the amountoftime that the read functions willwait before returningif thebusisidle.Ifaread functionis calledand therehasbeennonewdataonthebusforthe speci□edtimeout interval, the function will return with the `BG_READ_TIMEOUT`□ag of the `status`value

setandthe returnvaluewill indicatethenumberofbytesofdatathattheBeagleanalyzerwasable to capture prior to the timeout.If the timeout is set to `0`, there is no timeout interval and the read functions will block until

the requested amountof datais captured ora complete packet with the appropriatebus end

condition is observed.This setting is distinctly different than the latency setting. The timeout time should be set to avalue longer than the latency time.

Sleep (bg_sleep_ms)

```
u32 bg_sleep_ms (u32 milliseconds);
```

Sleepforgiven amountof time.

Arguments

`milliseconds`: number of milliseconds to sleep

ReturnValue

This function returns the number of milliseconds slept.

Speci□c Error Codes

None.

Details

This function provides a convenient cross–platform function to sleep the current thread using

standard operating system functions.The accuracy of this function depends on the operating system scheduler. This function willreturn the number of milliseconds that were actually slept.

TargetPower (bg_target_power)

```
int bg_target_power (Beagle beagle, u08 power_flag);
```

Activate/deactivate targetpower pins4and6.

Arguments

beagle: handle of a Beagle analyzerpower_mask: enumeratedvalues specifyingpowerpin state. SeeTable 6.

Table6:Power Flag de□nitions

BG_TARGET_POWER_OFF	Disable target power pin
BG_TARGET_POWER_ON	Enable target power pin
BG_TARGET_POWER_QUERY	Queries the target power pin state

ReturnValue

The current state of the target power pins on the Beagle analyzer will be returned. The con□g–uration willbe describedbythe samevalues asin the table above.

Speci□c Error Codes

BG_FUNCTION_NOT_AVAILABLE: The hardware version is not compatible with this feature. Only the BeagleI^{2}C/SPI/MDIO monitor supportsswitchable targetpower pins.

Details

This functionis onlyavailable on the BeagleI^{2}C/SPI/MDIO Protocol Analyzer.

Both target power pins are controlled together. Independent control is not supported. This function maybe executed in anyoperation mode. For the most part, target power should be left off, as the Beagle analyzer is normally passively

monitoring thebus.

Host Interface Speed (bg_host_ifce_speed)

```
int bg_host_ifce_speed (Beagle beagle);
```

Querythe host interface speed.

Arguments

beagle: handle of a Beagle analyzer

ReturnValue

This function returns enumerated values specifying the USB speed at which the analysis com−puteris communicatingwiththegivenBeagleanalyzer.SeeTable 7.

Table7:Interface Speed de□nitions

BG_HOST_IFCE_FULL_SPEED	Full-speed (12Mbps) interface
BG_HOST_IFCE_HIGH_SPEED	High-speed (480Mbps) interface

Speci□c Error Codes

None.

Details

Used to determine the USB communication rate between the Beagle analyzer and the analysis PC. The Beagle analyzers require a high−speed USB connection with the host. Capturing from a Beagle analyzer that is connected at full−speed can cause data to be lost and corruption of capture data.

Buffering

Host Buffer Size (bg_host_buffer_size)

```
int bg_host_buffer_size(Beagle beagle, u32 size_bytes);
```

Setthe amountofbuffering thatistobe allocatedonthe analysisPC

Arguments

beagle: handle of a Beagle analyzer

num_bytes: numberofbytesinbuffer

ReturnValue

This function returns the actual amountofbuffering set.

Speci□c Error Codes

BG_STILL_ACTIVE: An attempt was made to change the con□guration while the capture was still active.

Details

This function sets the amount of memory allocated tobuffering data that has been siphoned off the Beagle analyzerby the host software library,but notyet readby the application. The absolute minimumand maximumvaluesfor thisbuffersize are64kBand16MB, respectively. The requestedbuffer size is matched as closely as possibleby the function, and the function willkeepthe actualbuffersize within these boundaries. Forexample,if32kBofbufferingis requested, then 64 kB will actually be set.

If num_bytesis0,the functionwill returnthe amountofbuffering currentlysetonthePCand will leave the amountofbuffering unmodi□ed. This function canbe calledin thisfashioneven when the capture is active as it does not attempt to change the con□guration. It is important to note that bg_latency() and bg_sample_rate() can have an effect on the totalbuffer size. Therefore,to accurately determinehowmuchbufferinghasbeensetonthePC,thiscall should be made after all the con□gurations have been set.

If the application does not read data from the software libraryquickly enough, the entire host-sidebufferwill□ll.FormostoftheBeagleanalyzersthis meansthatanynewtraf□conthetarget bus will be dropped. The Beagle USB 480 analyzer, however, has a large on-board memory buffertosolvethisissue.To understandtheoperationoftheBeagleUSB480analyzerandhow it relates to the API, please refer to Section 6.8.

Available Read Buffering (bg_host_buffer_free)

```
int bg_host_buffer_free (Beagle beagle);
```

Querythe amountof readbufferingavailable.

Arguments

beagle: handle of a Beagle analyzer

ReturnValue

The amountofavailable USB readbufferinginbytes.

Speci□c Error Codes

None.

Details

USB read buffers are used by the analysis computer to receive the incoming data from the Beagle analyzer. Calling this function will returnthe amountofPCbufferingavailableto receive data as of the last bg_*_read() call.Ifthe amountofavailableUSBbufferingdrops closeto zero, capture data from the device maybe lost.

Used Read Buffering (bg_host_buffer_used)

```
int bg_host_buffer_used (Beagle beagle);
```

Querythe amountof used USB readbuffering.

Arguments

beagle: handle of a Beagle analyzer.

ReturnValue

The amountof used USB readbufferinginbytes.

Speci□c Error Codes

None.

Details

USB read buffers are used by the analysis computer to receive the incoming data from the Beagle analyzer. Calling this function will returnthe amountofPCbuffering filled with received data as of the last bg_*_read() call. If the amount of used USBbuffering comes close to the totalbuffer size, capture data from thedevice maybe lost.

Communication Speed Benchmark (bg_commtest)

```
int bg_commtest (Beagle beagle, int num_samples, int delay_count);
```

Test the Beagle analyzer communication link performance.

Arguments

beagle: handle of a Beagle analyzer num_samples: number of samples to receive from the analyzer.

delay_count: count delay on the host before processing each sample

ReturnValue

The number of communication errors received during the test.

Specific Error Codes

None.

Details

This function tests the host computer'sability to process data received from the Beagle analyzer. The function commands the given Beagle analyzer to send test packets at the given frequency (see bg_samplerate())to the host computer over the USB interface. Thedelay_countvari–able providesawayfor the application programmer to add an artificial counter delaybetween eachsample processedbythehost.Forlargedelay values,itwillbeharderforthehosttokeep up withthe datarateovertheUSBbus, therebyleadingto more communication errors.

Monitoring API

Enable Monitoring (bg_enable)

```
int bg_enable (Beagle beagle, BeagleProtocol protocol);
```

Start monitoring packets on the selected interface.

Arguments

beagle: handle of a Beagle analyzerprotocol: enumeratedvalues specifyingthe protocolto monitor(seeTable8)

*Table8:*BeagleProtocol*enumeratedvalues*

BG_PROTOCOL_NONE	No Protocol
BG_PROTOCOL_COMMTEST	Comm Tester

BG_PROTOCOL_USB	USB Protocol
BG_PROTOCOL_I2C	I2CProtocol
BG_PROTOCOL_SPI	SPI Protocol
BG_PROTOCOL_MDIO	MDIO Protocol

ReturnValue

ABeagle status code ofBG_OKis returned on success.

Speci□c Error Codes

BG_FUNCTION_NOT_AVAILABLE: The connected Beagle analyzer does not support capturing for the requested protocol.

BG_UNKNOWN_PROTOCOL: Aprotocol was requested that does not appear in the enumeration detailedinTable 8.

Details

This function enables monitoring on the given Beagle analyzer. See the section on the protocol−speci□c APIs. Functionsfor retrieving the capture data from the Beagle analyzer are described therein.

Stop Monitoring (bg_disable)

```
int bg_disable (Beagle beagle);
```

Stop monitoring of packets.

Arguments

beagle: handle of a Beagle analyzer

ReturnValue

ABeagle status code ofBG_OKis returned on success.

Speci□c Error Codes

None.

Details

Stops monitoring on the given Beagle analyzer.

Sample Rate (bg_samplerate)

```
int bg_samplerate (Beagle beagle, int samplerate_khz);
```

Set the sample rate in kilohertz.

Arguments

beagle: handle of a Beagle analyzersamplerate_khz: New sample rate in kilohertz

ReturnValue

This function returns the actual sample rate set.

Speci□c Error Codes

BG_FUNCTION_NOT_AVAILABLE: The Beagle analyzer does not supportchanging the sample rate.

BG_STILL_ACTIVE: An attempt was made to change the con□guration while the capture was still active.

Details

Changes the sampleratefora Beagle analyzer. Thedevicemust not currentlyhave monitoring enabled. If samplerate_khzis 0, the function will return the sample rate currently set on the Beagle analyzer and the sample rate will be left unmodi□ed. The Beagle USB 12 analyzer and the Beagle USB 480 analyzer do not support changing the sample rate, so it will always return the current sample rate.

Bit Timing Size (bg_bit_timing_size)

```
int bg_bit_timing_size (BeagleProtocol protocol, int num_data_bytes);
```

Getthesizeofthe timing dataforthegiven protocoland datasize.

Arguments

protocol: enumeratedvalues specifying the protocolof the data (seeTable 8)

num_data_bytes: numberof databytesexpected

ReturnValue

Thenumberof timing entriestoexpectforgivennumberof databytesforthegivenprotocol.

Speci□c Error Codes

None.

Details

Call this function before calling the bg_***_read_bit_timing() API functions to determine how large a bit_timingarrayto allocate.For BG_PROTOCOL_MDIO, this function will always return the value 32, regardless of the thevalue passedfor num_data_bytes.

6.5 Notes on Protocol-Speci□c Read Functions

All read functions returna statusvalue through the statusparameter.Table 9providesa listing of all the status codes that are shared throughout all the protocols.

Table9:Read Status de□nitions

BG_READ_OK	Read successful.

BG_READ_TIMEOUT	No data was seen before the timeout interval oc–curred. This mayindicate that no data was seen on the bus or there was a pause in the transmis–sion of data longer than the timeout interval.
BG_READ_ERR_MIDDLE_OF_PACKET	Data collection was started in the middle of a packet. This indicates that a transaction was al–ready being transmitted across the bus when the read function was called.
BG_READ_ERR_SHORT_BUFFER	The packet was longer than the buffer size. The buffer passed to the read function was too short to contain the full data of the transaction.
BG_READ_ERR_PARTIAL_LAST_BYTE	The last byte in the buffer is incomplete. The num–ber of bits of data captured did not align to the ex–pected data size. For example, for I2Cthe number of bits received was not a multiple of 9(8 data bits plus 1ACK/NACK bit).
BG_READ_ERR_UNEXPECTED	An unexpected event occurred on the bus. The event is still presented to the user, however it is tagged with this status □ag.

6.6 I^{2}CAPI Notes

The I^{2}CAPI functions are onlyavailablefor the BeagleI^{2}C/SPI/MDIO Protocol Analyzer.

I^{2}CMonitor Interface

I^{2}CPullups (bg_i2c_pullup)

```
int bg_i2c_pullup (Beagle beagle,
                   u08 pullup_flag);
```

Enables, disables and queries theI^{2}Cpullup resistors.

Arguments

beagle: handle of a Beagle analyzer pullup_flag: the functiontoperformas detailedinTable 10

Table10:Pullup de□nitions

BG_I2C_PULLUP_OFF	Disable the pullup resistors.
BG_I2C_PULLUP_ON	Enable the pullup resistors.
BG_I2C_PULLUP_QUERY	Querythe status of the pullup resistors.

ReturnValue

ABeagle status code ofBG_OKis returnedon success.Ifthevalue passedfor pullup_flagis BG_I2C_PULLUP_QUERY, the state of the pullups is returned.

Speci□c Error Codes

BG_FUNCTION_NOT_AVAILABLE: The hardware version is not compatible with this feature. OnlyI^{2}Cdevices supportswitchable pullup pins.

Details

Sets and queries the state of the pullup resistors on theI^{2}Clines. Normally the pullups will be setbythe host and targetdevices, so this function will notbe used.

ReadI^{2}C(bg_i2c_read)

```
int bg_i2c_read (Beagle beagle,
                 u32 status,
                   *
                 u64 time_sop,
                   * u64 time_duration,
                   * u32 time_dataoffset,
                   *
                 int max_bytes,
                 u16 data_in);
                   *
```

Read packet from theI^{2}Cport.

Arguments

beagle: handle of a Beagle analyzer

status: □lled with the status bitmask as detailedinTables 9and11 time_sop: □lled with the timestamp when the packet begins time_duration: □lled with the number of ticks that it took to read the data time_dataoffset: □lled with the timestamp when data appeared on thebus max_bytes: maximumnumberofbytesto read data_in: an allocated arrayof u16which is □lled with the received data

Table 11: I^{2}CSpeci□c Read Status de□nitions

BG_READ_I2C_NO_STOP The I^{2}C stop condition was not observed on the bus. This canbe caused eitherbya read timeout orby aI^{2}Crepeated start condition.

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci□c Error Codes

None.

Details

Thefunction willblockuntilthe requested amountof datais captured,a complete packet witha stopor repeated startconditionis observed,orthebusis idlefor longer thanthe timeout interval set. See Section 6.4for information on thebg_latency () and bg_timeout () functions which affect the behavior of this function.

For each u16 written todata_inbythe function,thelower 8−bits representthevalueofabyte of data sent across thebus and bit8represents theACK orNACKvaluefor thatbyte.A0in bit 8representsanACKanda1inbit8representsaNACK.SeeTable12for constants thatmay be used as bit mask to access the appropriate □elds in data_in.

All of the timing data is measured in ticks of the sample rate clock.

Table 12: I²CData Mask constants

Constant name	Value	Description
BG_I2C_MONITOR_DATA	0x00ff	Mask to access data □eld.
BG_I2C_MONITOR_NACK	0x0100	Mask to access ACK/NACK □eld.

The data_inpointer should be allocated at least as large as max_bytes. All of the timing data is measured in ticks of the sample clock.

ReadI²Cwith data−level timing (bg_i2c_read_data_timing)

```
int bg_i2c_read_data_timing (Beagle beagle,
                             u32 status,

                                 * u64 time_sop,

                                 * u64 time_duration,

                                 * u32 time_dataoffset,

                                 * int max_bytes,
```



```
                             u16 data_in,

                                 * int max_timing, u32
                             data_timing);

                                 *
```

Read data from theI²Cport.

Arguments

common_args: see bg_i2c_read () for common arguments max_timing: size of data_timingarray

data_timing: an allocated array of u32which is □lled with timing data for each data word read

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci□c Error Codes

None.

Details

This function is an extension ofthe `bg_i2c_read()` function with the addedfeature of giving data—level timing. All of the `bg_i2c_read()` arguments and details apply.The values in the `data_timing`array give the offset of the start of each data word from`time_sop`.Adataword includes all8bitsof data aswell as the acknowledgment bit.

The `data_timing`arrayshould be allocated at least as large as `max_timing`.

ReadI[2]Cwith bit—level timing (bg_i2c_read_bit_timing)

```
int bg_i2c_read_bit_timing (Beagle beagle,
                            u32 * status,
                            u64 time_sop,

                                * u64 time_duration,

                                * u32 time_dataoffset,

                                * int max_bytes, u16
                            data_in,

                                * int max_timing, u32
                            bit_timing);

                                *
```

Read data from theI[2]Cport.

Arguments

`common_args`: see `bg_i2c_read()` for common arguments `max_timing`: size of `bit_timing`array `bit_timing`: an allocated arrayof `u32`whichis □lled with the timing datafor each bit read

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci□c Error Codes

None.

Details

This function is an extension ofthe `bg_i2c_read()` function with the addedfeature of givingbit—level timing. All of the `bg_i2c_read()` arguments and details apply.

The values in the `bit_timing`arraygive the offset of each bit from `time_sop`.The `bit_timing`array should be allocated at least as large as `max_timing`. Use the func—tion `bg_bit_timing_size()` (in Section 6.4)to determine how large an array to allocatefor
bit_timing.

Notes

The SPI API functions are onlyavailablefor the BeagleI[2]C/SPI/MDIO Protocol Analyzer.

SPI Monitor Interface

SPI Con□guration (bg_spi_con□gure)

```
int bg_spi_configure (Beagle beagle, BeagleSpiSSPolarity ss_polarity,
                      BeagleSpiSckSamplingEdge sck_sampling_edge,
                      BeagleSpiBitorder bitorder);
```

Sets SPIbus parameters.

Arguments

beagle: handle of a Beagle analyzer ss_polarity: sets the slave select detection to active−low or active−high bit polarity, seeTa−ble13
sck_sampling_edge: sets data sampling on the leading or trailing edge of the clock signal, seeTable 14 bitorder: sets big−endian or little−endian bit order, seeTable 15

Table 13: *SPI SS Polarity definitions*

BG_SPI_SS_ACTIVE_LOW	Set active low polarity
BG_SPI_SS_ACTIVE_HIGH	Set active high polarity

Table 14: *SPI SCK Sampling Edge definitions*

BG_SPI_SCK_SAMPLING_EDGE_RISING	Sample on the leading edge
BG_SPI_SCK_SAMPLING_EDGE_FALLING	Sample on the trailing edge

Table15:SPI Bit Order de□nitions

BG_SPI_BITORDER_MSB	Big-endian bit ordering
BG_SPI_BITORDER_LSB	Little-endian bit ordering

ReturnValue

ABeagle statuscodeofBG_OKis returnedon successoran errorcodeas detailedinTable30.

Speci□c Error Codes

BG_STILL_ACTIVE: An attempt was made to change the con□guration while the capture was still active.

BG_FUNCTION_NOT_AVAILABLE: The hardware version is not compatible with this feature. Only the I^{2}C/SPI/MDIO device supports SPI con□guration.

Details

The SPI standard is much more loosely de□ned than I^{2}C, MDIO, or USB. As a consequence, the SPI monitor must be con□gured to match the parameters of the device being monitored. If the con□guration of the SPI monitor does not match the con□guration of the SPI devices being monitored, the capture data from the monitor may be corrupted.

Read SPI (bg_spi_read)

```
int bg_spi_read (Beagle beagle,
                 u32 status,

                   * u64 * time_sop, u64
                 time_duration,

                   * u32 time_dataoffset,

                   * int mosi_max_bytes, u08
                 data_mosi,

                   * int miso_max_bytes, u08
                 data_miso);

                   *
```

Read data from the SPI port.

Arguments

beagle: handle of a Beagle analyzer status: □lled with the status bitmask as detailed in Table 9 time_sop: □lled with the timestamp when the data read begins time_duration: □lled with the number of ticks that it took to read the data time_dataoffset: □lled with the timestamp when data appeared on the bus mosi_max_bytes: maximum number of MOSI bytes to □ll data_mosi: an allocated array of u08 which is □lled with the data sent from the master to the slave miso_max_bytes: maximum number of MISO bytes to □ll data_miso: an allocated array of u08 which is □lled with the data sent from the slave to the master

Return Value

This function returns the number of bytes read or a negative value indicating an error.

Speci□c Error Codes

None.

Details

The function will block until the requested amount of data is captured, a complete packet with slave select deassertion is observed, or the bus is idle for longer than the timeout interval set. See Section 6.4

for information on the bg_latency () and bg_timeout () functions which affect the behavior of this function.

The data_mosi array should be allocated at least as large as mosi_max_bytes. The data_miso array should be allocated at least as large as miso_max_bytes.

All of the timing data is measured in ticks of the sample clock.

Read SPI with data-level timing (bg_spi_read_data_timing)

```
int bg_spi_read_data_timing (Beagle beagle,
                             u32 status,
                             * u64 time_sop,
                             * u64 time_duration,
                             * u32 time_dataoffset,
                             * int mosi_max_bytes, u08
                             data_mosi,
                             * int miso_max_bytes, u08
                             data_miso,
                             * int max_timing, u32
                             data_timing);
                             *
```

Read data from the SPI port.

Arguments

common_args: see bg_spi_read () for common arguments max_timing: size of data_timing array

data_timing: an allocated array of u32 which is lled with timing data for each data word read

ReturnValue

This function returns the number of bytes read or a negative value indicating an error.

Specic Error Codes

None.

Details

This function is an extension of the bg_spi_read () function with the added feature of byte-level timing. All of the bg_spi_read () arguments and details apply. The values in the data_timing array give the offset of the start of each data word from time_sop. For SPI, a data word is considered a single byte.

The data_timing array should be allocated at least as large as max_timing.

Read SPI with bit–level timing (bg_spi_read_bit_timing)

```
int bg_spi_read_bit_timing (Beagle beagle,
                            u32 status,

                                 * u64 time_sop,

                                 * u64 time_duration,

                                 * u32 * time_dataoffset, int
                            mosi_max_bytes, u08 data_mosi,

                                 * int miso_max_bytes, u08
                            data_miso,

                                 * int max_timing, u32
                            bit_timing);

                                 *
```


Read data from the SPI port.

Arguments

common_args: see bg_spi_read() for common arguments max_timing: size of bit_timingarray bit_timing: an allocated arrayof u32whichis □lled with the timing datafor each bit read

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci□c Error Codes

None.

Details

This function is an extension of the bg_spi_read() function with the addedfeatureof bit–leveltiming. All of the bg_spi_read() arguments and details apply.

The values in the bit_timingarraygive the offset of each bit from time_sop.The bit_timingarray should be allocated at least as large as max_timing. Use the func–tion bg_bit_timing_size() (in Section 6.4)to determine how large an array to allocatefor bit_timing.

TOTAL PHASE

6.8 USB API

Notes

1. The USB12 API functions are onlyavailablefor the Beagle USB12 Protocol Analyzer.
2. The USB 480 API functions are onlyavailablefor the Beagle USB 480 Protocol Analyzer.
3. The □rst byte of the captured USB packet is the packet ID (PID). An enumeration is provided that de□nes all the possible PIDs whichis listedinTable 16.

Table16:USBPacket ID de□nitions

BG_USB_PID_OUT	0xe1
BG_USB_PID_IN	0x69
BG_USB_PID_SOF	0xa5
BG_USB_PID_SETUP	0x2d
BG_USB_PID_DATA0	0xc3
BG_USB_PID_DATA1	0x4b
BG_USB_PID_DATA2	0x87
BG_USB_PID_MDATA	0x0f
BG_USB_PID_ACK	0xd2
BG_USB_PID_NAK	0x5a
BG_USB_PID_STALL	0x1e
BG_USB_PID_NYET	0x96
BG_USB_PID_PRE	0x3c
BG_USB_PID_ERR	0x3c
BG_USB_PID_SPLIT	0x78
BG_USB_PID_PING	0xb4
BG_USB_PID_EXT	0xf0

4. In additiontothegeneralread statusvaluesinTable 9,theUSBread functionscanalso return USB speci□c statusvalues. The enumerated types are listedinTable 17.

5. Additional event informationis returnedbythe USB read functions through the `events`argument. The event information is bitmask encoded with the enumerated types de□ned in Table 18. Refer to Section 1.1 for details on how these events pertain to the USB architecture.

Table17:USB Read Status de nitions

Status Codes for USB 12 and USB 480	
BG_READ_USB_ERR_BAD_SIGNALS	Incorrect line states
BG_READ_USB_ERR_BAD_PID	Captured packet has bad PID
BG_READ_USB_ERR_BAD_CRC	Captured packet has bad CRC
USB 12 Speci c Status Codes	
BG_READ_USB_ERR_BAD_SYNC	Cannot nd SYNC signal
BG_READ_USB_ERR_BIT_STUFF	Bit stuf ng error detected
BG_READ_USB_ERR_FALSE_EOP	Incorrect End of packet
BG_READ_USB_ERR_LONG_EOP	End of packet too long
USB 480 Speci c Status Codes	
BG_READ_USB_TRUNCATION_MODE	Captured packet in truncation mode
BG_READ_USB_END_OF_CAPTURE	Capture has ended

Table18:USB Event Code de nitions

Event Codes for USB 12 and USB 480	
BG_READ_USB_HOST_DISCONNECT	Target Host disconnected
BG_READ_USB_TARGET_DISCONNECT	Target Device disconnected
BG_READ_USB_HOST_CONNECT	Target Host connected
BG_READ_USB_TARGET_CONNECT	Target Device connected
BG_READ_USB_RESET	Bus put into reset state
USB 480 Speci c Event Codes	
BG_EVENT_USB_J_CHIRP	Chirp-J detected
BG_EVENT_USB_K_CHIRP	Chirp-K detected
BG_EVENT_USB_SPEED_UNKNOWN	Communication speed is unknown
BG_EVENT_USB_LOW_SPEED	Low-speed bus operation detected
BG_EVENT_USB_FULL_SPEED	Full-speed bus operation detected
BG_EVENT_USB_HIGH_SPEED	High-speed bus operation detected
BG_EVENT_USB_LOW_OVER_FULL_SPEED	Low-over-full-speed bus operation detected
BG_EVENT_USB_SUSPEND	Bus has entered suspend state
BG_EVENT_USB_RESUME	Bus has left suspend state
BG_EVENT_USB_KEEP_ALIVE	Low-speed keep-alive detected
BG_EVENT_USB_OTG_HNP	OTG HNP detected
BG_EVENT_USB_OTG_SRP_DATA_PULSE	OTG SRP data-line pulse detected
BG_EVENT_USB_OTG_SRP_VBUS_PULSE	OTG SRP Vbus pulse detected
BG_EVENT_USB_DIGITAL_INPUT	One or more digital inputs have changed state
BG_EVENT_USB_DIGITAL_INPUT_MASK	Bitmask of line state for each input pin

Read USB (bg_usb12_read)

```
int bg_usb12_read (Beagle beagle,
                   u32 status,

                       * u32 * events, u64
                   time_sop,

                       * u64 time_duration,

                       * u32 time_dataoffset,

                       * int max_bytes, u08
                   packet);

                       *
```

Read data from the USB port.

Arguments

beagle: handle of a Beagle analyzer status: □lled with the status bitmask as detailedinTable 9andTable 17 events: □lled with theevent bitmask as detailedinTable 18 time_sop: □lled with the timestamp when the data read begins time_duration: □lled with the number of ticks that it took to read the data time_dataoffset: □lled with the timestamp when data appeared on thebus max_bytes: maximumnumberofbytesto read packet: an allocated arrayof u08which is □lled with the received data

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci□c Error Codes

None.

Details

The function willblockuntil the requested amountof datais captured,a completepacket withthe appropriate end of packet condition is observed, or thebus is idlefor longer than the time–out interval set. See Section 6.4 for information on the bg_latency() and bg_timeout() functions which affect the behavior of this function.

The packetarrayshould be allocated at least as large as max_bytes.All of the timing data is measured in ticks of the sample clock. The Beagle USB 12 analyzer islocked to a 48 MHz sample rate, thus each count measures 20.83 ns.

Read USB with data–level timing (bg_usb12_read_data_timing)

```
int bg_usb12_read_data_timing (Beagle beagle,
                               u32 status,

                                   * u32 events,
```

```
* u64 time_sop,

* u64 * time_duration, u32
time_dataoffset,

*
```

www.pc17.com.cn
TOTAL PHASE

```
int max_bytes, u08 packet,

* int max_timing, u32
data_timing);

*
```

Read data from the USB port.

Arguments

common_args: see bg_usb12_read() for common arguments max_timing: size of data_timingarray

data_timing: an allocated array of u32which is □lled with timing data for each data-word read

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci□c Error Codes

None.

Details

This function is an extension of the bg_usb12_read() function with the addedfeatureofbyte-level timing. All of the bg_usb12_read() arguments and details apply.The values in the data_timingarray give the offset of the start of each data word fromtime_sop.For USB,a datawordis considereda singlebyte.

The data_timingarrayshould be allocated at least as large as max_timing.

Read USB with bit-level timing bg_usb12_read_bit_timing)

```
int bg_usb12_read_bit_timing (Beagle beagle,
                              u32 status,

                                * u32 events,

                                * u64 time_sop,

                                * u64 time_duration,

                                * u32 time_dataoffset,

                                * int max_bytes, u08 packet,

                                * int max_timing, u32
                              bit_timing);
```

*

Read data from the USB port.

Arguments

common_args: see bg_usb12_read() for common arguments max_timing: size of bit_timingarray

bit_timing: an allocated arrayof u32whichis □lled with the timing datafor each bit read

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci□c Error Codes

TOTAL PHASE

None.

Details

This function is an extension of the bg_usb12_read() function with the addedfeature of bit-level timing. All of the bg_usb12_read() arguments and details apply.

The values in the bit_timingarraygive the offset of each bit from time_sop.The bit_timingarray should be allocated at least as large as max_timing. Use the func-tion bg_bit_timing_size() (in Section 6.4)to determine how large an array to allocatefor

bit_timing.

Con□gure USB 480 Capture (bg_usb480_capture_con□gure)

```
int bg_usb480_capture_configure (Beagle beagle, BeagleUsb480CaptureMode capture_mode,
                                                         * BeagleUsb480TargetSpeed
                                 target_speed);
                                                         *
```

Con□gure the Beagle USB 480 analyzer.

Arguments

beagle: handle of a Beagle analyzer capture_mode: modeof packet capture as detailedinTable 19

target_speed: intended speedof packet capture as detailedinTable 20

*Table19:*BeagleUsb480CaptureMode*enumeratedvalues*

BG_USB480_CAPTURE_REALTIME	Con□gure to real–time capture
BG_USB480_CAPTURE_REALTIME_WITH_PROTECTION	Con□gure to real–time capture with over□ow protection
BG_USB480_CAPTURE_DELAYED_DOWNLOAD	Con□gure to delayed–download mode

*Table20:*BeagleUsb480TargetSpeed*enumeratedvalues*

BG_USB480_AUTO_SPEED_DETECT	Con□gure to auto–detect the bus speed
BG_USB480_LOW_SPEED	Con□gure to lockto low–speed capture
BG_USB480_FULL_SPEED	Con□gure to lockto full–speed capture
BG_USB480_HIGH_SPEED	Con□gure to lockto high–speed capture

ReturnValue

ABeagle statuscodeofBG_OKis returnedon successoran errorcodeas detailedinTable30.

Speci□c Error Codes

BG_STILL_ACTIVE: An attempt was made to change the con□guration while the capture was still active.

Details

These con□guration parameters specify the speed and capture mode of the Beagle USB 480 analyzer.The capture_modeoption speci□es whether the capture will be in real–time, real–time withtruncation, or delayed–download mode. For more details on the different modes of capture,refer to Section 3.3.

The target_speedoption speci□esthespeedofcommunicationonthetargetbus.TheBeagleUSB 480 Analyzer maybe con□gured to auto–detect the speed, or mayalternatively be lockedto monitor only a single communication speed.

TOTAL PHASE

Enable Digital Output (bg_usb480_digital_out_con□g)

```
int bg_usb480_digital_out_config (Beagle beagle,
                                  u08 out_enable_mask,
                                  u08 out_polarity_mask);
```

Enable Beagle analyzer to output a speci□c match type on output pins.

Arguments

beagle: handle of a Beagle analyzer out_enable_mask: bitmaskof enabled output pins as detailedinTable 21 out_polarity_mask: bitmaskof polarity on outputs pins as detailedinTable 22

Table21:Digital Output Pin Enable bit mask

BG_USB480_DIGITAL_OUT_ENABLE_PIN1	Enables Output Pin 1
BG_USB480_DIGITAL_OUT_ENABLE_PIN2	Enables Output Pin 2
BG_USB480_DIGITAL_OUT_ENABLE_PIN3	Enables Output Pin 3
BG_USB480_DIGITAL_OUT_ENABLE_PIN4	Enables Output Pin 4

Table22:Digital Output PinPolarity bit mask

BG_USB480_DIGITAL_OUT_PIN1_ACTIVE_HIGH BG_USB480_DIGITAL_OUT_PIN1_ACTIVE_LOW	Output Pin 1idles low Output Pin 1idles high
BG_USB480_DIGITAL_OUT_PIN2_ACTIVE_HIGH BG_USB480_DIGITAL_OUT_PIN2_ACTIVE_LOW	Output Pin 2idles low Output Pin 2idles high
BG_USB480_DIGITAL_OUT_PIN3_ACTIVE_HIGH BG_USB480_DIGITAL_OUT_PIN3_ACTIVE_LOW	Output Pin 3idles low Output Pin 3idles high
BG_USB480_DIGITAL_OUT_PIN4_ACTIVE_HIGH BG_USB480_DIGITAL_OUT_PIN4_ACTIVE_LOW	Output Pin 4idles low Output Pin 4idles high

ReturnValue

ABeagle statuscodeofBG_OKis returnedon successoran errorcodeas detailedinTable30.

Speci□c Error Codes

BG_CONFIG_ERROR: An attempt was made to set an invalid con□guration.

Details

Pins aretriggeredbyparticularevents which are detailedin Section 3.3. Pleasereferto Sec-tion 2.1for the hardware speci□cationsof the output pins.The out_enable_maskinput is a bitmask of the parameters listed in Table 21. By using abit-wiseORoperation,multiple outputpinscanbeenabled.Itisimportanttonotethat callingthis function will disable all pins that are not explicitly set in the out_enable_maskinput.

The out_polarity_maskinput con□gures the polarity of the output. Like out_enable_mask,this bitmask allows the user to con□gure multiple pins through a bit-wise OR operation. Thedefault con□guration is active low. If a pin is attempted to be con□gured as both active low andactive high, then it will only actually con□gure to active high.

Digital output lines will activate as soon as their triggering event is fully con□rmed.

TOTAL PHASE

MatchDigital Output (bg_usb480_digital_out_match)

```
int bg_usb480_digital_out_match (
                        Beagle beagle,
                        BeagleUsb480DigitalOutMatchPins pin_num,
                        BeagleUsb480PacketMatch packet_match,
                                                * BeagleUsb480DataMatch
                        data_match);
                                                *
```

Enable Beagle analyzerto output matchonaparticularbus data.

Arguments

beagle: handle of a Beagle analyzer pin_num: outputpinstobe enabledas detailedinTable 23 packet_match: USB packet header information and boolean operations that the Beagle ana-lyzer can match packet headers with data_match: USB packet data and boolean operations thatthe Beagle USB 480 analyzer can match incoming packet data with

ReturnValue

ABeagle statuscodeofBG_OKis returnedon successoran errorcodeas detailedinTable30.

Speci□c Error Codes

BG_STILL_ACTIVE: An attempt was made to change the con□guration while the capture was still active.

BG_CONFIG_ERROR: An attempt was made to set an invalid con□guration.

Details

The function is used to con□gure the output pins of the digital I/O port to trigger on speci□c events. This function shouldbe called repeatedlyfor eachpin thatmustbe con□gured. Output pins 1 and 2 do not use the packet_matchand data_matchinputs, as they do not require that extra information. They are therefore completely con□gurable from the bg_usb480_out_config() function and calling this function on either of those pins will re-turn BG_CONFIG_ERROR.

Output pin 3 does not use the data_matchinput because it does not have that functional-ity. Therefore, calling this function with a non-NULL value in data_matchwill also return BG_CONFIG_ERROR.

The BeagleUsb480PacketMatchand BeagleUsb480DataMatchmust be used to correctly con□gure the matching capabilitiesof Output Pins3and4.The BeagleUsb480PacketMatchstructure describes the packet parameters that need to bematched.

Table 23: BeagleUsb480DigitalOutMatchPins *enumerated values*

BG_USB480_DIGITAL_OUT_MATCH_PIN3	Selects Output Pin 3
BG_USB480_DIGITAL_OUT_MATCH_PIN4	Selects Output Pin 4

/* Digital ouput matching configuration */ struct BeagleUsb480PacketMatch {


```
    BeagleUsb480MatchType pid_match_type;
    u08 pid_match_val;
    BeagleUsb480MatchType dev_match_type;
    u08 dev_match_val;
    BeagleUsb480MatchType ep_match_type;
    u08 ep_match_val;

};
```

The BeagleUsb480DataMatchstructure describes the data sequence that need to be matched.

```
struct BeagleUsb480DataMatch {
    BeagleUsb480MatchType data_match_type;
    u08 data_match_pid;
    u16 data_length;
    u08 data;


        *
    u16 data_valid_length;
    u08 data_valid;


        *
};
```

The BeagleUsb480MatchTypeenumerated type is used throughout the two structures to de-termine whether the match should assert on the values being equal, not equal, or don't care (disabled). The different enumerated types are describedin thefollowing table.

*Table24:*BeagleUsb480MatchType*enumeratedvalues*

BG_USB480_MATCH_TYPE_DISABLED	The match type is disabled
BG_USB480_MATCH_TYPE_EQUAL	The match type must equal
BG_USB480_MATCH_TYPE_NOT_EQUAL	The match type must not equal

The BeagleUsb480DataMatchstructure has its own □eld for checking PIDs. This □eld is a bitmaskfor eachof thefour typesof data packets andis describedin thefollowing table.

Table25:Data Match PID bit mask

BG_USB480_DATA_MATCH_DATA0	Enable match on data with DATA0PID
BG_USB480_DATA_MATCH_DATA1	Enable match on data with DATA1PID
BG_USB480_DATA_MATCH_DATA2	Enable match on data with DATA2PID

BG_USB480_DATA_MATCH_MDATA	Enable match on data with MDATAPID

Since the BeagleUsb480DataMatchhas itsown □eldsfor matching the PID,using the structure will therefore overwrite the PID settings de□ned in BeagleUsb480PacketMatch. Furthermore, the data matching is determined through two arrays. The dataarraydetermines which values the userwouldliketo match.The □rstbyteofthisarray would correlatetothe□rstbyteofthe packet. The second array, data_valid, determines which of thosebytes in the dataarray arevalidfor matching. Settingabytetozeroin the data_validarray means that byte is a don't-care conditionfor the matching algorithm.

Thedigital outputsactivateas soonastheirtriggeringeventcanbefully con□rmed. Thus,Pins1 and2will activate as soon as the capture activates or rxactive goes high, respectively. However,

Pins3and4must assureamatchofalloftheircharacteristics. Therefore,onlyonceallpossible PIDs, device address, and endpoints of a given packet are checked completely can the output activate.The assertionof matcheddataonPin4 mustwaituntiltheendofthedatapacketto assureamatch.Packetsthatareshorterthenwhatis de□nedbythe BeagleUsb480DataMatchstructuremaystillactivatePin4ifallthedatauptothatpoint matched correctly.

Enable USB 480 Digital Input (bg_usb480_digital_in_con□g)

```
int bg_usb480_digital_in_config (Beagle beagle,
                                 u08 in_enable_mask);
```

Enablesthe analyzerto sendaneventon changestotheexternal inputsonthe DigitalI/Oport.

Arguments

beagle: handle of a Beagle analyzer in_enable_mask: bitmaskof enabled input pins as detailedinTable 26

Table26:Digital Input Pin Enable bit mask

BG_USB480_DIGITAL_IN_ENABLE_PIN1	Enable input pin 1
BG_USB480_DIGITAL_IN_ENABLE_PIN2	Enable input pin 2
BG_USB480_DIGITAL_IN_ENABLE_PIN3	Enable input pin 3
BG_USB480_DIGITAL_IN_ENABLE_PIN4	Enable input pin 4

ReturnValue

ABeagle statuscodeofBG_OKis returnedon successoran errorcodeas detailedinTable30.

Speci□c Error Codes

None.

Details

The Beagle USB 480 analyzer digital I/O port hasfour pins allocatedfor digital inputs. These digital inputs will display events in-line with collected data. For further details on the digital inputs refer to Section 2.1 and Section 3.3.

The in_enable_maskisa bitmaskof the parameters listedinTable 26.By usinga bit-wiseOR operation, multiple input pins can be enabled. It is important to note that calling this function will disable all pins that are not explicitly set in the enable_maskinput.

Enable Hardware Filter (bg_usb480_hw_filter_config)

```
int bg_usb480_hw_filter_config (Beagle beagle,
                                u08 filter_enable_mask};
```

Specify hardware filtering modes.

Arguments

beagle: handle of a Beagle analyzer filter_enable_mask: hardware filtering configuration definitions as detailedinTable 27

ReturnValue

Table27:Hardware Filter Enable bit mask

BG_USB480_HW_FILTER_PID_SOF	Filter SOFpackets
BG_USB480_HW_FILTER_PID_IN	Filter IN+ACKIN+NAKpacket groups
BG_USB480_HW_FILTER_PID_PING	Filter PING+NAKpacket groups
BG_USB480_HW_FILTER_PID_PRE	Filter PREpacket groups
BG_USB480_HW_FILTER_PID_SPLIT	Filter SPLITpacket groups
BG_USB480_HW_FILTER_SELF	Filter packets intended for Beagle analyzer

ABeagle statuscodeofBG_OKis returnedon successoran errorcodeas detailedinTable30.

Specific Error Codes

None.

Details

The Beagle USB 480 Analyzer is capable of filtering out data-less transactions before being savedfor capture. This option can be especially usefulfor saving memory on the analysis PC and on the hardwarebuffer.

To enable the filtering, simply use the bitmask detailed in Table 27. By using a bit-wise OR operation, multiple filters can be enabled. It is important to note that calling this function will disable all filters that are not explicitly set in the filter_configinput.

For more detailed information on the hardware filters, please refer to Section3.3.

USB Buffer Statistics (bg_usb480_hw_buffer_stats)

```
int bg_usb480_hw_buffer_stats (Beagle beagle,
                               u32 buffer_size,
```

```
* u32 buffer_usage,
* u08 buffer_full);
*
```

Outputs real time statisticsfor the on–boardbuffer.

Arguments

beagle: handle of a Beagle analyzerbuffer_size: total sizeof the hardwarebufferbuffer_usage: amountof space usedin the hardwarebufferbuffer_full: indicates whether thebufferis full

ReturnValue

ABeagle statuscodeofBG_OKis returnedon successoran errorcodeas detailedinTable30.

Speci□c Error Codes

None.

Details

The function returns up–to–date statistical information about the on–board hardwarebuffer. This is especially useful for delayed–download captures to poll the status of the buffer. However, calling this function issues a short communication between the Beagle USB 480 analyzer and theanalysisPC.IftheBeagleanalyzerisonthesamebusthatitis monitoring,thencallstothis

function will take upbus bandwidth and can take up on–board memoryspace due to the USB broadcast architecture(see Section 1.1).Ifbus bandwidthisa concern,then pollingthebuffer shouldbekepttoa minimum. If pollingis required, thenitis recommended that Self Filtering be enabled in order to eliminate the packets intendedfor the Beagle analyzer, and thus save on–board memory.

Read USB (bg_usb480_read)

```
int bg_usb480_read (Beagle beagle,
                    u32 status,

                        * u32 events,

                        * u64 time_sop,

                        * u64 * time_duration, u32
                    time_dataoffset,

                        * int max_bytes, u08
                    packet);

                        *
```

Read data from USB port.

Arguments

beagle: handle of a Beagle analyzer status: □lled with status bitmask as detailedinTable 9andTable 17 events: □lled withevent bitmask as detailedinTable 18 time_sop: timestamp when the data read begins time_duration: number of ticks that it took to read the data time_dataoffset: timestamp when data appeared on thebus max_bytes: maximumnumberofbytesto read packet: arrayofbytes whichis lled with the received data

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci c Error Codes

None.

Details

The function willblockuntil the requested amountof datais captured,a completepacket with the appropriate end of packet condition is observed, or thebus is idlefor longer than the time–out interval set. See Section 6.4 for information on the bg_latency () and bg_timeout () functions which affect the behavior of this function.

The packetarrayshould be allocated at least as large as max_bytes. All of the

timing data is measured in ticks of the sample rate clock.

The rstbyteof the USB packetis the packetID.An enumerationis provided that de nes all the possible packet IDsinTable 16.

In additiontothegeneralreadstatusvaluesinTable 9,therearesomeUSBspeci cstatusval–uesenumeratedinTable 17. The user shouldbeawareofthe BG_READ_USB_END_OF_CAPTURE

status code, this is speci c to the Beagle USB 480 analyzer and will be returned if the bg_usb480_read () function is called after a capture has completed.

The eventsenumeration describes speci c events that have occurred during the USB cap–ture. By masking the eventsvalue with the ones detailed inTable 18, the user can determine whether a speci c event has occurred.

It should also be noted that if a packet is returned when in truncated mode, the packet length willbelimitedto4bytes.The functionwillstillreturnthetruelengthofthepacket,howeveronly uptothe rst4bytesofdatawillbe insertedintothe packetarray. The remainingbytes willbe lled with 0s.

Also,theuseofdigitalinputsmay causecertainbuseventstoappearoutoforder.SeeSection 3.3for more information.

Reconstruct Bit Timing (bg_usb480_reconstruct_timing)

```
int bg_usb480_reconstruct_timing (BeagleUsb480TargetSpeed speed,
                                  int num_bytes,
                                  u08 packet,

                                          *
                                  int max_timing,
                                  u32 bit_timing);
```

*

Reconstruct the bit–level timing of a packet.

Arguments

`speed`: thebus speedof the packet `num_bytes`:numberofbytestodothe reconstructionon `packet`: an arraycontaining the packetbytes `max_timing`: maximum number of bits to do the reconstruction on `bit_timing`: allocated arrayof `u32`which is □lled with the duration of each of the bits

ReturnValue

ABeagle statuscodeofBG_OKis returnedon successoran errorcodeas detailedinTable30.

Speci□c Error Codes

None.

Details

The Beagle USB 480 analyzer is restricted to packet–level timing of the capture data. However, this function provides a bit–level timing reconstruction based upon the data and the speed of the bus.

The `bit_timing`arraywill be □lled with the duration of each of the bits in the `packet`array. The duration of each bit is provided in counts of a 480 MHz clock, corresponding to approximately a2 nsresolution. Those bits that arefollowedby a bit–stuff willhaveaduration thatis twiceas long asa normal bit timefor that speed.

The `bit_timing`array should be allocated at least as large as `max_timing`. Use the func–tion `bg_bit_timing_size()` (in Section 6.4)to determine how large an array to allocatefor `bit_timing`.

Notes

The MDIO API functions are onlyavailablefor the BeagleI[2]C/SPI/MDIO Protocol Analyzer.

MDIO Monitor Interface

Read MDIO (bg_mdio_read)

```
int bg_mdio_read (Beagle beagle,
                  u32 status,

                      * u64 time_sop,

                      * u64 time_duration,

                      * u32 time_dataoffset,

                      * u32 data_in);

                      *
```

Read data from the MDIO port.

Arguments

beagle: handle of a Beagle analyzer status: □lled with the status bitmask as detailedinTable 9 time_sop: □lled with the timestamp when the frame preamble begins time_duration: □lled with the number of ticks that from time_sopto the last bit of the MDIO

frame time_dataoffset: □lledwiththenumberofticksfromtime_sopuntil the end of the preamble data_in: a pointer to a u32value which is □lled with the received MDIO data

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci□c Error Codes

None.

Details

The functionwillblock untila completeframeis capturedorthebusisidleforlongerthanthe timeout interval set. See Section 6.4for information on thebg_latency () and bg_timeout () functions which affect the behavior of this function.

All of the timing data is measured in ticks of the sample clock.

Read MDIO with bit–level timing (bg_mdio_read_bit_timing)

```
int bg_mdio_read_bit_timing (Beagle beagle,
                             u32 * status,
                             u64 time_sop,

                                 * u64 time_duration,

                                 * u32 time_dataoffset,

                                 * u32 data_in

                                 * int max_timing, u32
                             bit_timing);

                                     *
```


Read data from the MDIO port.

Arguments

common_args: see bg_mdio_read () for common arguments max_timing: size of bit_timingarray bit_timing: an allocated arrayof u32whichis □lled with the timing datafor each bit read

ReturnValue

This function returns thenumberofbytes read ora negativevalue indicating an error.

Speci□c Error Codes

None.

Details

This functionisanextensionofthe bg_mdio_read () function withthe addedfeatureof bit–leveltiming. All of the bg_mdio_read () arguments and details apply.

The values in the bit_timingarraygive the offset of each bit from time_sop.The bit_timingarray should be allocated at least as large as max_timing. Use the func–tion bg_bit_timing_size () (in Section 6.4)to determine how large an array to allocatefor

bit_timing.

Thebit timeforthe □nalbitoftheframeisalwayszero. Thisisduetothefact thatthebit timesare measured between rising edges of the MDC line. The □rst bit time is measured from the□rst rising edge of the MDC line to the next rising edge. For the last bit of a frame, there maynot be a subsequent rising edge of the MDC line until the next frame. Therefore, no bit timevalue canbe determinedfor □nal bitofaframe.

Parse MDIO data (bg_mdio_parse)

```
int bg_mdio_parse (u32 packet,
                   u08 clause,

                       *
                   u08 opcode,

                       *
                   u08 addr1,

                       *
                   u08 addr2,

                       *
                   u16 data);

                       *
```

Parses packet into □eld values.

Arguments

packet: the MDIO frame to parse clause: □lled with the clauseof theframe as detailedinTable 28 opcode: □lled with theOP codeof theframe as detailedinTable 29 addr1: □lled with the value of the □rst address □eld (PHY in Clause 22, port in Clause 45) addr2: □lled with the value of the second address □eld (reg in Clause 22, device in Clause 45) data: □lled with the contentsof the data portionof theframe

ReturnValue

ABeagle statuscodeofBG_OKis returnedon successoran errorcodeas detailedinTable30.

Speci□c Error Codes

Table28:MDIO Clause de□nitions

BG_MDIO_CLAUSE_22	0x00	MDIO Clause 22
BG_MDIO_CLAUSE_45	0x01	MDIO Clause 45
BG_MDIO_CLAUSE_ERROR	0x02	Unknown value in clause □eld

Table29:MDIO Opcode de□nitions

BG_MDIO_OPCODE22_WRITE	0x01	Clause 22 write OP code
BG_MDIO_OPCODE22_READ	0x02	Clause 22 read OP code
BG_MDIO_OPCODE22_ERROR	0xff	Clause 22 unknown OP code
BG_MDIO_OPCODE45_ADDR	0x00	Clause 45 address OP code
BG_MDIO_OPCODE45_WRITE	0x01	Clause 45 write OP code
BG_MDIO_OPCODE45_READ_POSTINC	0x02	Clause 45 post read increment ad-dress OP code
BG_MDIO_OPCODE45_READ	0x03	Clause 45 read OP code

BG_MDIO_BAD_TURNAROUND: An unexpected value in turnaround □eld of the frame.

Details

The return value will indicate validity of the turnaround □eld. BG_OKindicates the value of the turnaround □eld is valid. BG_MDIO_BAD_TURNAROUNDindicates an invalid value in the turnaround □eld.

6.10 Error Codes

Table30:Beagle API Error Codes

Literal Name	Value	bg_status_string() return value
BG_OK	0	ok
BG_UNABLE_TO_LOAD_LIBRARY	−1	unable to load library
BG_UNABLE_TO_LOAD_DRIVER	−2	unable to load usb driver
BG_UNABLE_TO_LOAD_FUNCTION	−3	unable to load function
BG_INCOMPATIBLE_LIBRARY	−4	incompatible library version
BG_INCOMPATIBLE_DEVICE	−5	incompatible device version
BG_INCOMPATIBLE_DRIVER	−6	incompatible driver version
BG_COMMUNICATION_ERROR	−7	communication error
BG_UNABLE_TO_OPEN	−8	unable to open device
BG_UNABLE_TO_CLOSE	−9	unable to close device
BG_INVALID_HANDLE	−10	invalid device handle
BG_CONFIG_ERROR	−11	con□guration error
BG_UNKNOWN_PROTOCOL	−12	unknown beagle protocol
BG_STILL_ACTIVE	−13	beagle still active
BG_FUNCTION_NOT_AVAILABLE	−14	beagle function not available
BG_COMMTEST_NOT_AVAILABLE	−100	comm test feature not available
BG_COMMTEST_NOT_ENABLED	−101	comm test not enabled
BG_I2C_NOT_AVAILABLE	−200	i2c feature not available
BG_I2C_NOT_ENABLED	−201	i2c not enabled
BG_SPI_NOT_AVAILABLE	−300	spi feature not available
BG_SPI_NOT_ENABLED	−301	spi not enabled
BG_USB_NOT_AVAILABLE	−400	usb feature not available
BG_USB_NOT_ENABLED	−401	usb not enabled
BG_MDIO_NOT_AVAILABLE	−500	mdio feature not available
BG_MDIO_NOT_ENABLED	−501	mdio not enabled
BG_MDIO_BAD_TURNAROUND	−502	mdio bad turnaround □eld

7 Legal/Contact

7.1 Disclaimer

Allof the software and documentation providedinthis datasheet,is copyrightTotal Phase, Inc. ("Total Phase"). License is granted to the user to freely use and distribute the software and documentationin completeand unalteredform,provided thatthepurposeisto useorevaluate Total Phase products. Distribution rights do not include public posting or mirroring on Internet websites.OnlyalinktotheTotalPhasedownload areacanbeprovidedonsuchpublicwebsites.

Total Phase shallinnoeventbe liabletoanypartyfor direct, indirect, special, general, inciden−tal, or consequential damages arising from the use of its site, the software or documentation downloaded from its site, or any derivative works thereof, even if Total Phase or distributors have been advised of the possibility of such damage. The software, its documentation, and any derivative works is provided on an “as–is” basis, and thus comes with absolutely no war–ranty, eitherexpressor implied. This disclaimer includes,butisnot limitedto, impliedwarranties of merchantability, □tnessfor any particular purpose, and non–infringement. Total Phase and distributors have no obligation to provide maintenance, support, or updates.

Information in this document is subject to change without notice andshould not be construed as acommitmentbyTotal Phase. Whiletheinformation contained hereinis believedtobe accurate, Total Phase assumes no responsibilityfor anyerrors and/or omissions that mayappear in this document.

7.2 Life Support EquipmentPolicy

Total Phase products are not authorizedfor usein life supportdevices or systems. Life support devices or systems include, but are not limited to, surgical implants, medical systems, and other safety–critical systems in which failure of a Total Phase product could cause personal injury or lossof life. ShouldaTotal Phase productbe usedin such an unauthorized manner, Buyer agrees to indemnify and hold harmless Total Phase, its of□cers, employees, af□liates, and distributors from anyand all claims arising from such use, even if such claim alleges that Total Phase was negligent in the design or manufacture of its product.

7.3 Contact Information

Total Phase can befound on the Internet at http://www.pc17.com.cn/. Ifyou have support–related questions, please email the product engineers at support@pc17.com.cn.For sales inquiries, please contact sales@pc17.com.cn.

List of Figures

List of Tables

TOTAL PHASE

Contents

TOTAL PHASE

TOTAL PHASE