

MBARI PUCK Specification

Version 1.3

Monterey Bay Aquarium Research Institute
Draft date: June 17, 2009

Document maintained by Tom O'Reilly (oreilly@mbari.org)

Document change log

January 28, 2006: Clarified instrument mode and PUCK mode definitions in sections 1.25, 1.26, and 3.2

February 6, 2006: Clarified use of PUCKFM command in sections 3.2.3, 3.3.3

March 7, 2006: Clarified response to PUCK soft break in section 3.2.2; clarified meaning of datasheet UUID in section 4.2.1.

October 2, 2008: Removed leading <CR> from PUCKRDY response to PUCKSA command in section 3.3.6

May 15, 2009: Added reference to Leach-Salz UUID in section 4.2.1.

June 17, 2009: Clarified processing of soft break when device is already in PUCK mode, in section 3.2.2; updated state diagram (Figure 1) accordingly.

Contents

- 1 Overview
 - 1.1 Description
 - 1.2 Terms and Definitions
 - 1.3 Document Conventions
- 2 PUCK Requirements
 - 2.1 Common PUCK Requirements
 - 2.2 External PUCK Requirements
- 3 PUCK Protocol
 - 3.1 Serial Interface
 - 3.2 PUCK State Transitions and Command Processing
 - 3.3 Command Set
 - 3.4 Error Codes
- 4 PUCK Instrument Datasheet
 - 4.1 Instrument datasheet description
 - 4.2 Instrument datasheet entries
- 5 PUCK Physical Interface
 - 5.1 PUCK Interface
 - 5.2 External PUCK Interface

1 Overview

1.1 PUCK Description

Many instruments are designed to interact with a host computer through a serial interface. Various interactions are possible; the host may provide a user interface for the instrument, may log the instrument's data, or may distribute the data to a wider network. In order to perform these functions, the host requires information about the instrument, including serial port configuration, knowledge of commands recognized by the instrument, and metadata that describes the instrument and the science data it produces. Most instruments currently do not themselves supply all of this information automatically, but instead require that the host be configured beforehand; part of the configuration process generally involves installation of driver software on the host, which actually handles interaction with the instrument. Configuration usually involves several steps, and can be a time-consuming, tedious, and potentially error-prone process.

The PUCK (Programmable Underwater Connector, with Knowledge) addresses this issue by enabling automatic system configuration when the instrument is plugged into the host. The PUCK is a small device that provides persistent storage of instrument configuration information, and interfaces that allow the information to be retrieved by a host. The PUCK may be permanently externally attached to the serial interface of an instrument or the PUCK protocol may be embedded within an existing or new instrument. During deployment, when the PUCK-enabled instrument is plugged into the host, the host can retrieve and utilize the information from the PUCK.

1.2 Terms and Definitions

1.2.1 PUCK, PUCK-enabled device – a device that recognizes the MBARI PUCK protocol. The term “PUCK” can mean either “external PUCK” or “embedded PUCK” (see below).

1.2.2 PUCK specification – defines how information is stored and retrieved from a PUCK-enabled device; includes a protocol definition and stored information formats

1.2.3 Host platform – a controller that is capable of communicating with attached serial instruments

1.2.4 Instrument – a data-gathering device that may be composed of one or more sensors sharing a serial interface

1.2.5 PUCK mode – the state of a PUCK-enabled device in which PUCK serial commands are recognized and processed. Response to commands outside of PUCK protocol while in PUCK mode is not defined by this specification, and in fact “native” instrument commands could be recognized and processed while in PUCK mode without violating the PUCK specification.

1.2.6 Instrument mode - the state of a PUCK-enabled device in which "native" instrument commands are recognized and processed. Note that PUCK commands may also be recognized while the device is in instrument mode, without violating the PUCK specification.

1.2.7 Metadata – Information about the data that enables data processing, or that puts the data into useful context. A description of the instrument and its data format are examples of metadata.

1.2.8 Plug-and-Work – Automated integration of an instrument into an observing system that occurs when the instrument is physically plugged into the system.

1.2.9 Instrument datasheet – Information that describes the PUCK payload and the associated instrument; every PUCK contains an instrument datasheet. The PUCK instrument datasheet has a standard format defined by the PUCK specification.

1.2.10 PUCK payload – Optional information that can be stored in a PUCK; the payload format is not defined by the PUCK specification, but can be defined by individual observing systems.

1.2.11 External PUCK – a small device that can be attached to any serial instrument and provide PUCK functionality.

1.2.12 Embedded PUCK – a serial instrument that implements the PUCK protocol in addition to its native instrument command set.

1.3 Document Conventions

The following are conventions that are used through out this document.

1.3.1 Serial input/output is displayed in the `courier` font, for example

```
GB<CR>  
19200<CR>PUCKRDY<CR>
```

1.3.2 The notation `<CR>` and `<LF>` are used to denote carriage return and linefeed respectively.

1.3.3 The notation `[0,255]` is used to denote a parameter range (e.g. $0 \leq \text{parameter} \leq 255$)

Unless otherwise noted, the following requirements, formats, and protocols apply to external and embedded PUCKs.

2 PUCK Requirements

2.1 PUCK common Requirements

The PUCK common requirements are applicable to embedded as well as external PUCK implementations.

2.1.1 The PUCK shall store information in the persistent storage in a flat binary format.

2.1.2 The PUCK protocol shall allow the amount of persistent storage on the PUCK to be scaled to the needs of the application.

2.1.3 The PUCK shall provide no less than 96 bytes of memory (instrument datasheet plus payload?)

2.1.4 The PUCK shall be ready to process commands from the host platform within 500 milliseconds of PUCK device power up.

2.1.5 The PUCK shall support a RS-232 serial interface.

2.1.6 The PUCK shall configure its serial interface to 8 data bits, no parity, and 1 stop bit.

2.2 External PUCK Requirements

The external PUCK requirements apply only to an external PUCK implementation.

2.2.1 The external puck shall be able to route all serial traffic between host and instrument.

2.2.2 The external PUCK shall operate within the same environmental range as the attached instrument.

2.2.3 The external PUCK shall operate with in the voltage range provided by the host platform.

3 PUCK Protocol

3.1 PUCK Serial Interface

The PUCK serial interface uses a simple command response protocol. When in PUCK mode, the PUCK will interpret a group of ASCII characters beginning with "PUCK" and terminated by a <CR> as a PUCK command. All PUCK commands are case-sensitive, and are in upper-case. When a command is issued the PUCK will attempt to execute that command. Upon successful execution the PUCK will return the characters

```
PUCKRDY<CR>
```

This indicates that the PUCK is ready for the next command. The string "PUCK<CR>" is also a valid command (the "null command") that results in a PUCKRDY response. If the PUCK is unable to execute a command beginning with the characters "PUCK" successfully it will issue the characters

```
ERR #####<CR>  
PUCKRDY<CR>
```

Where the '####' is a four digit base ten error code in the range [1,9999]. If the command is a request for information then the command will return the information requested followed by the characters

```
PUCKRDY<CR>
```

The PUCK will terminate any transaction initiated by a PUCK command with the characters

```
PUCKRDY<CR>
```

at which point it is ready for another command to be issued.

3.2 PUCK State Transitions, Command Processing, and Command Sequences

Figure 1 depicts states and state-transitions of a PUCK-enabled instrument, from the standpoint of a host. While the PUCK-enabled device is in PUCK mode, PUCK commands will be processed by the device; during this mode, the host platform can retrieve information from the PUCK using the PUCK command protocol. Following retrieval of information from the PUCK, the host can issue a command to put the device into instrument mode. When the device enters instrument mode the host platform can communicate with it using instrument-specific commands. A device will automatically transition from PUCK mode to instrument mode after 2 minutes of command inactivity. A PUCK-enabled device can be switched from instrument mode to PUCK mode via the "PUCK soft break" mechanism.

As shown in the state diagram, a PUCK-enabled device will be in instrument mode following a full power cycle. At other times, the host can use the mechanisms shown in the diagram to determine the state of the device.

Note that the definitions of instrument mode and PUCK mode (sections 1.2.5 and 1.2.6) allow implementations that respond to both PUCK and instrument-specific commands within a single mode, without violating any provisions of this specification.

3.2.1 PUCK timeout

If a PUCK-enabled device implementation does not recognize instrument commands while in PUCK mode, it shall automatically switch from PUCK mode to instrument mode following 2

minutes of inactivity, as measured from the *completion* of the last issued PUCK command; thus the device shall never transition to instrument mode while a PUCK command is executing. The device shall write the string "PUCKTMO<CR>" to its serial port when PUCK mode timeout occurs. Implementations that always recognize instrument commands while in PUCK mode need not implement PUCK timeout.

3.2.2 PUCK soft break

A host may switch a PUCK-enabled device from instrument mode to PUCK mode by issuing a *PUCK soft break*, which consists of the following sequence:

```
"@@@@@@"  
(wait 750 milliseconds)  
"!!!!!"  
(wait 500 milliseconds)
```

(A soft break received by a device already in PUCK mode shall be treated as a successful command.)

Note that a host may not necessarily "know" the baud rate for which the device UART is configured; thus the host may need to issue the soft break at several baud rates. The host can confirm a successful soft break by sending the null PUCK command ("PUCK<cr>") to the device; a response of PUCKRDY indicates that the device is now in PUCK mode. Note that the device must not switch its baud rate in response to a soft break.

3.2.3 Writing to the PUCK

In the following sections that describe individual PUCK commands, reference is made to a "write session". Note that the sequence of commands used when writing a PUCK are specified here to accommodate implementation in resource constrained environments and using available FLASH memory technology. Thus a "write session" is defined as the following sequence of commands, with no other commands intervening:

1. The PUCKEM command to erase all of PUCK memory
2. Optionally a PUCKSA command to set the starting address
3. One or more PUCKWMM commands to write the PUCK contents
4. The PUCKFM command to flush PUCK contents to non-volatile storage

Note that this implies that the PUCK memory must be written contiguously. Note also that the PUCKFM (flush PUCK memory) command should be used only after a write session is complete; i.e. after PUCKFM is issued, no more PUCKWMM commands are allowed until a new write session is initiated with PUCKEM.

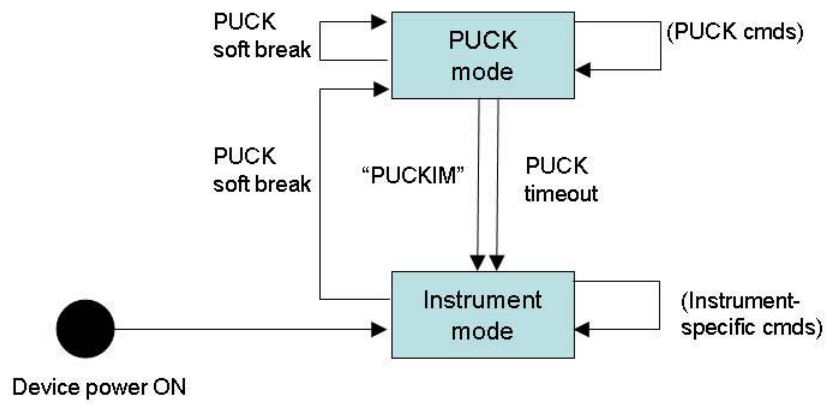


Figure 1: PUCK state diagram

3.3 PUCK Command Set

Table 1 – PUCK Command Summary

Command	Description
PUCKRM	Read from PUCK memory
PUCKWM	Write to PUCK memory
PUCKFM	End PUCK write session
PUCKEM	Erase PUCK memory
PUCKGA	Get address of PUCK internal memory pointer
PUCKIM	Put PUCK into instrument mode
PUCKSA	Set address of PUCK internal memory pointer
PUCKSZ	Get the size of the PUCK memory
PUCKSB	Set PUCK baud rate
PUCKTY	Query PUCK type
PUCKVB	Verify baud rate support
PUCKVR	Get PUCK protocol version string
PUCK	Null command

The following subsections discuss each command in detail. Note that for each command there is an associated “timeout” specification. Timeout is defined as the time between transmission of the command’s terminating carriage return and receipt of the first byte of the device response message.

3.3.1 PUCKRM – Read from PUCK memory

Description

The PUCKRM command is used to read bytes from the PUCK memory. The command shall accept an ASCII parameter from 0 to 1024 specifying the number of bytes requested. The read memory command shall respond with the ASCII character '[' followed by the number of binary bytes requested starting at the byte pointed to by the memory address pointer (see commands PUCKSA and PUCKGA for memory address pointer details). If the memory pointer reaches the end of PUCK memory it shall rollover to address 0 and continue returning bytes until the number of bytes requested has been sent. When the final byte has been sent the PUCK shall append the byte packet with the ASCII character ']' and the PUCKRDY<CR> terminator.

Errors

ERR 0020 – the number of bytes requested is out of range.

Timeout: 500 milliseconds

Example

```
PUCK RX: PUCKRM 10<CR>
PUCK TX: [0123456789] PUCKRDY<CR>
PUCK RX:
PUCK TX:
```

3.3.2 PUCKWM – Write to PUCK memory

Description

The PUCKWM command is used to write bytes to the PUCK memory. The command shall accept an ASCII parameter from 0 to 32 specifying the number of bytes to be written to PUCK memory, expressed in decimal format. After the PUCKWM command and parameter have been entered the PUCK shall accept the number of binary bytes requested for storage to PUCK memory. The bytes shall be stored starting at the address pointed to by the memory pointer (see commands PUCKSA and PUCKGA for memory pointer details). The memory pointer shall be incremented appropriately for each successful PUCKWM command executed by the PUCK. If the number bytes requested to write exceeds the PUCKS capacity based on the current address of the memory pointer then an ERR 0021 shall be returned. If the PUCK attempts to write to a read-only region of memory an ERR 0022 shall be returned (see section 4.1 for description of the read-only instrument datasheet option). If a write is attempted to a memory location that has not been initialized by a previous PUCKEM command, an ERR 0023 is returned.

Errors

ERR 0020 – the number of bytes to be written is out of range.

ERR 0021 – address requested out of range.

ERR 0022 – write attempted to read only memory.

ERR 0023 – write attempted to non-initialized memory

Timeout: 500 milliseconds

Example

```
PUCK RX: PUCKWM 10<CR>0123456789
PUCK TX: PUCKRDY<CR>
```

PUCK RX:
PUCK TX:

3.3.3 PUCKFM – End PUCK write session

Description

The PUCKFM command is used to insure that any bytes buffered by the puck during a write session (section 3.2.3) are stored to PUCK memory immediately. Following a PUCKFM command, no other PUCKWM commands should be attempted until a new write session is initiated with PUCKEM. A PUCK shall accept the PUCKFM command whether or not the particular PUCK implementation requires it.

Errors

Timeout: 30 seconds

Example

```
PUCK RX: PUCKFM<CR>
PUCK TX: PUCKRDY<CR>
PUCK RX:
PUCK TX:
```

3.3.4 PUCKEM – Erase PUCK memory

Description

The PUCKEM command is used to erase all of the PUCK memory with the exception of any read-only sections. The command shall put the memory in a state such that the PUCKWM command can be used to store bytes to PUCK memory. Any write sessions (section 3.2.3) to the PUCK should be preceded by the PUCKEM command. After the execution of the PUCKEM command the address of the PUCK memory pointer (see commands PUCKSA and PUCKGA for memory pointer details) will be 0.

Errors

Timeout: 30 seconds

Example

```
PUCK RX: PUCKEM<CR>
PUCK TX: PUCKRDY<CR>
PUCK RX:
PUCK TX:
```

3.3.5 PUCKGA – Get address pointed to by the PUCK internal memory pointer

Description

The PUCKGA command is used to return the memory address (expressed in decimal) pointed to by the PUCK's internal memory pointer. When the PUCKRM or PUCKWM command are used the PUCK reads or writes bytes starting at the address pointed to by the memory pointer. For every byte that is read or written to the PUCK the memory pointer will be incremented by 1.

Errors

Timeout: 500 milliseconds

Example

PUCK RX: PUCKGA<CR>

PUCK TX: 1234<CR>PUCKRDY<CR>

PUCK RX:

PUCK TX:

3.3.8 PUCKIM – Put PUCK into instrument mode

Description

The PUCKIM command is used to connect the serial lines to the attached instrument. Once in instrument mode, all serial traffic from the host is passed through the PUCK to the instrument, and all serial traffic from the instrument is passed to the host. An embedded PUCK shall accept the PUCKIM command whether or not it is necessary for the instrument.

Errors

Timeout: 500 milliseconds

Example

```
PUCK RX: PUCKIM<CR>
PUCK TX:
PUCK RX:
PUCK TX:
```

3.3.6 PUCKSA – Set address pointed to by the PUCK internal memory pointer

Description

The PUCKSA command is used to set the memory address pointed to by the PUCK'S internal memory pointer. When the PUCKRM or PUCKWM command are used the PUCK reads or writes bytes starting at the address pointed to by the memory pointer. For every byte that is read or written to the PUCK the memory pointer will be incremented by 1. If an attempt is made to set the memory pointer to an address outside of the PUCK memory range the PUCK shall indicate that an address range violation has occurred. The specified address is expressed in decimal format.

Errors

ERR 0021 – memory address out of range

Example

```
PUCK RX: PUCKSA 1234<CR>
PUCK TX: PUCKRDY<CR>
PUCK RX:
PUCK TX:
```

3.3.7 PUCKSZ – Get the size of the PUCK memory

Description

The PUCKSZ command is used to determine the total number of bytes that may be stored in PUCK memory. PUCKSZ returns the total (used and unused) number of bytes expressed in decimal format.

Errors

Timeout: 500 milliseconds

Example

```
PUCK RX: PUCKSZ<CR>
PUCK TX: 1048576<CR>PUCKRDY<CR>
PUCK RX:
PUCK TX:
```

3.3.9 PUCKSB – Set PUCK baud rate

Description

The PUCKSB command is used to change the PUCK baud rate. The PUCKSB command followed by an ASCII representation of a valid baud rate shall cause the PUCK change to the requested baud rate. The PUCK shall return the PUCKRDY<CR> prompt at the newly requested baud rate if successful. If the requested baud rate is not available then the puck shall return an error code indicating that the requested baud rate is invalid for this implementation of the PUCK. The host can determine if a particular baud rate is valid by using the PUCKVB command.

Errors

Timeout: 500 milliseconds

ERR 0010 – Invalid baud rate requested

Example

```
PUCK RX: PUCKSB 19200<CR>
PUCK TX: PUCKRDY<CR>
PUCK RX:
PUCK TX:
```

3.3.10 PUCKTY – Query PUCK type

Description

The PUCKTY command is used to query the PUCK and determine the PUCK type. A Hex ASCII value will be returned containing the sum of various PUCK type flags.

Table 2 – PUCK type flags

PUCK Attribute	Description
0001	Read only datasheet memory
0002	PUCK hardware is external to the instrument
0004 – 8000	Reserved

Errors

Timeout: 500 milliseconds

Example

```
PUCK RX: PUCKTY<CR>
PUCK TX:          0003<CR>PUCKRDY<CR>
PUCK RX:
PUCK TX:
```

3.3.11 PUCKVB – Verify baud rate support

Description

The PUCKVB command is used to verify a PUCK implementation supports a particular baud rate. The VB command followed by an ASCII representation of a baud rate shall cause the PUCK to return YES<CR> or NO<CR> followed by the PUCKRDY<CR> prompt.

Errors

Timeout: 500 milliseconds

Example

```
PUCK RX: PUCKVB 9600<CR>
PUCK TX:          YES<CR>PUCKRDY<CR>
PUCK RX: PUCKVB 1234<CR>
PUCK TX:          NO<CR>PUCKRDY<CR>
```

3.3.12 PUCKVR – Get PUCK protocol version string

Description

The PUCKVR command is used to request a version identifier of the PUCK implementation. When the PUCKVR command is issued the PUCK shall return a version string having the following format:

vx.y

where the literal lowercase “v” is followed by “major version number” x, followed by literal ‘.’, followed by the “minor version number” y. There are no spaces in the version string. The values of x and y are determined by the version of MBARI PUCK protocol which has been implemented.

Errors

Timeout: 500 milliseconds

Example

```
PUCK RX: PUCKVR<CR>
PUCK TX:          v1.3<CR>PUCKRDY<CR>
PUCK RX:
PUCK TX:
```

3.3.13 PUCK – Null PUCK command

Description

The PUCK command simply results in a PUCKRDY response from the device if it is in PUCK mode.

Errors

Timeout: 100 milliseconds

Example

```
PUCK RX: PUCK<CR>
PUCK TX:          PUCKRDY<CR>
PUCK RX:
PUCK TX:
```

3.4 PUCK Error Codes

Table 3 – PUCK Error Codes

Error Code	Description
0004	No command match
0010	Invalid baud rate requested
0020	Bytes requested for read or write out of range
0021	Address requested out of range
0022	Write attempted to read only memory
0023	Write attempted to non-initialized memory

4 PUCK Instrument Datasheet

4.1 Instrument datasheet description

The PUCK instrument datasheet shall occupy the first 96 bytes of storage. The instrument datasheet may optionally be read-only. In the case of embedded PUCKs it is desirable to have a read-only instrument datasheet, as the information should never change for the life of the instrument. It can be determined whether an instrument datasheet is read-only via the PUCKTY command. The datasheet entries shall be stored in the following formats,

U32 – an unsigned 32-bit integer stored in big endian format
U16 – an unsigned 16-bit integer stored in big endian format
UUID – the Leach-Salz variant of a universally unique identifier
CHAR ARRAY – an array of ASCII characters

Summary of instrument datasheet entries

Table 4 – Instrument Datasheet Summary

Description	Size (bytes)	Format
UUID for instrument	16	UUID
Version of instrument datasheet	2	U16
Datasheet size	2	U16
Manufacture ID	4	U32
Manufacture model	2	U16
Manufacture version	2	U16
Serial number	4	U32
Instrument name	64	CHAR ARRAY
Total size	96	

4.2 Instrument datasheet entries

4.2.1 UUID – Universally unique identifier for the instrument

The UUID shall be a Leach-Salz variant of a universally unique identifier that is assigned to the instrument that is associated with this PUCK. The UUID uniquely identifies a specific *instance* of an instrument; thus two different instruments having identical manufacturer, model, and version codes will have different UUIDs. A description of the Leach-Salz UUID generation algorithm can be found at <http://www.ietf.org/rfc/rfc4122.txt>.

4.2.2 Version – Instrument datasheet version

The version shall be a U16 number identifying the version of the MBARI PUCK Specification that defines the instrument datasheet structure. All unassigned version numbers are reserved for future use.

Table 5 – Instrument Datasheet versions

Instrument datasheet version	Specification revision
1	MBARI PUCK Specification revision 1.2
2	MBARI PUCK Specification revision 1.3

4.2.3 Datasheet size – instrument datasheet size

The datasheet size entry shall be a U16 number specifying the size in bytes of the instrument datasheet, expressed in decimal format.

4.2.4 Manufacture ID – Identifier of instrument manufacture

The manufacture identifier shall be a U32 number that identifies a manufacture that has registered with the Marine Plug-and-Work consortium.

Table 6 – Manufacture ID numbers

Manufacture ID	Description
0	No manufacture ID assigned
1 – 255	Experimental use
256 – 4,294,967,295	Managed by Marine Plug-and-Work consortium

4.2.5 Manufacture Model – The model of a manufactures instrument

The manufacture model is a U16 that shall be used by the manufacture to identify different instrument models. A value of 0 means that no model has been assigned to this instrument. All assigned model numbers shall be made available by the instrument manufacture for use by plug-and-work application developers.

4.2.6 Manufacture Version – The version of a manufactures instrument model

The manufacture version is a U16 that shall be used by the manufacture to differentiate between different versions of the same model instrument. A value of 0 means that no version has been assigned to this instrument. All assigned version numbers shall be made available by the instrument manufacture for use by plug-and-work application developers.

4.2.7 Serial Number – Instrument serial number

The serial number is a U32 number that shall be assigned by manufactures. The serial number shall be set to 0 if it is not available.

4.2.8 Instrument Name – ASCII string containing instrument name

The instrument name is a free-form ASCII string containing the name of the instrument. Any unused characters should be set to 0. If the instrument name is not used all characters should be set to 0.

5 PUCK Physical Interface

5.1 PUCK Interface

All PUCK implementations shall support a subset of the standard RS-232 serial interface signals and power as shown in table 7. The PUCK or instrument electronics may be implemented with isolated communications and power returns if required by the application, but it is not necessary for the PUCK protocol to work on a particular host platform.

Table 7 – PUCK Interface Description

Signal	Description
RX	Serial port receive line
TX	Serial port transmit line
power	Instrument and puck power
return	Signal return

5.2 External PUCK Interface

The PUCK electronics may be implemented with isolated communications and power returns if an instrument requires it, but it is not necessary for a PUCK to work on a particular host platform.

External PUCK Instrument Interface

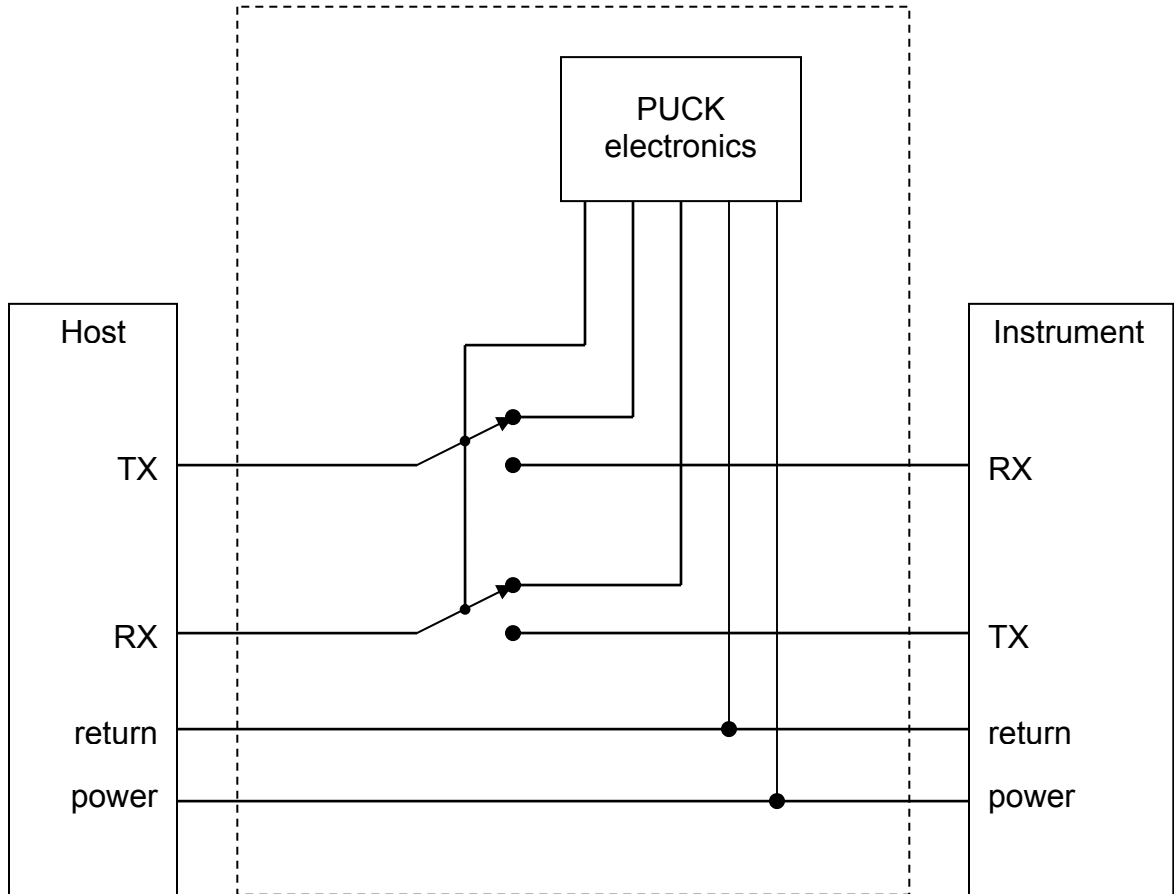


Figure 2