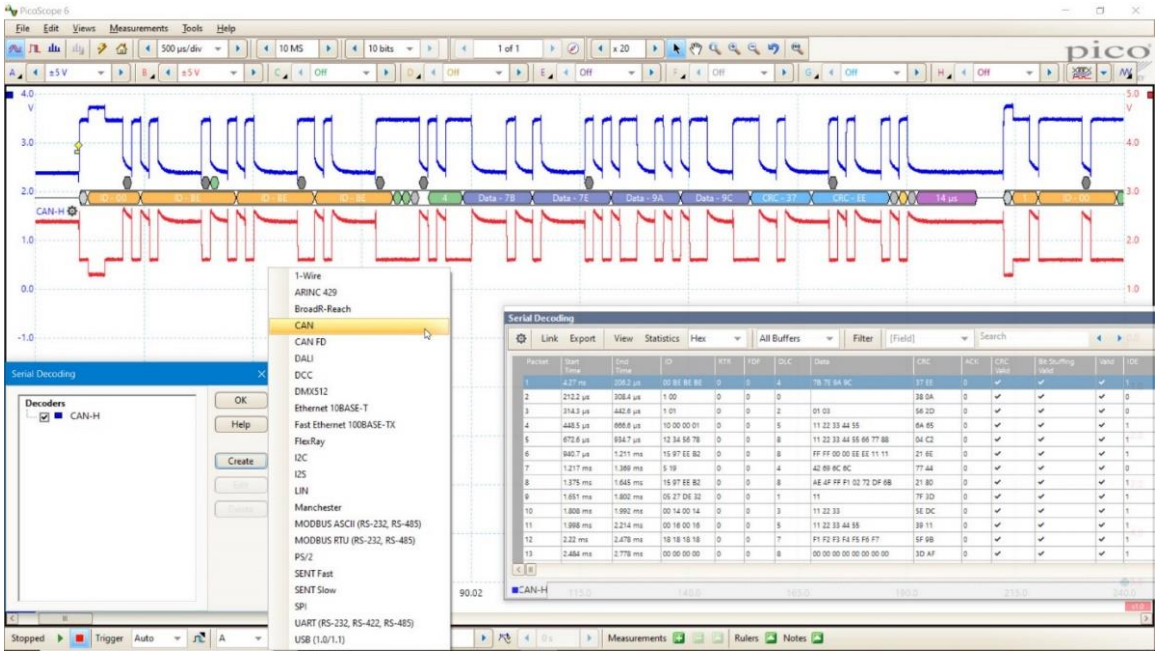
Reference Guide

Serial Bus Decoding with PicoScope



Introduction

Serial communication buses are used extensively in modern electronic designs. Serial buses offer significant cost advantages and some performance improvements over parallel bus communications. First off, there are fewer signals to route on the board, so PCB costs are lower. Less I/O pins on each device are needed, which simplifies component packaging and so reduces component cost. Some serial buses use differential signalling which improves noise immunity.

There is a wide range of serial communication standards, each optimised for specific operating conditions and differing design complexity, different speeds, power consumption, fault tolerance and, of course, cost.

Although serial buses offer several advantages, they also present difficulties when troubleshooting and debugging systems since the data is transmitted in packets or frames that need to be decoded, according to the standard in use, before the designer can make sense of the information flow. Manually decoding (or “bit counting”) streams of binary data is error prone and time-consuming.

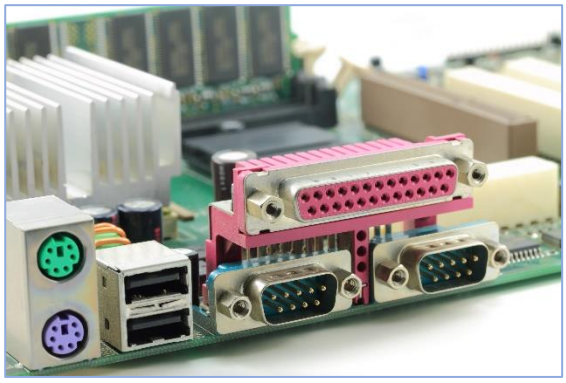
PicoScope includes decoding and analysis of popular serial standards to help engineers see what is happening in their design to identify programming and timing errors and check for other signal integrity issues. Timing analysis tools help to show performance of each design element, enabling the engineer to identify those parts of the design that need to be improved to optimize overall system performance.

PicoScope can decode 1-Wire, ARINC 429, BroadR-Reach (100BASE-T1), CAN, DALI, DCC, DMX512, Ethernet 10Base-T and 100Base-TX, FlexRay, I²C, I²S, LIN, Manchester, PS/2, SENT, SPI, UART (RS-232 / RS-422 / RS-485), and USB protocol data as standard, with more protocols in development, and available in the future with free-of-charge software upgrades.

In this guide we look at some of the more common serial bus protocols and how to decode them:

Contents

RS-232/UART

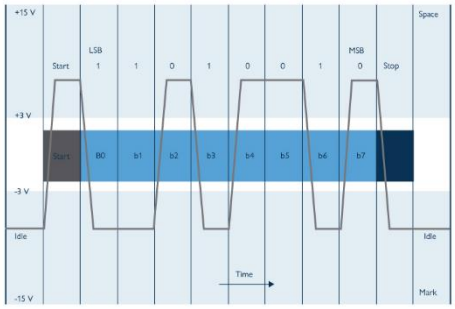


RS-232 is a standard for serial data communication first defined in 1962 by the Electronic Industries Alliance for use with data communication devices such as teletypewriters.

Later, personal computers and other devices made use of the RS-232 standard for connection to peripheral devices such as modems, mice, keyboards, etc.

In recent years RS-232 has become increasingly displaced by USB in modern PCs, though the standard, and many variants, are still widely used in industrial machines, networking equipment and scientific instruments.

RS-232 signaling



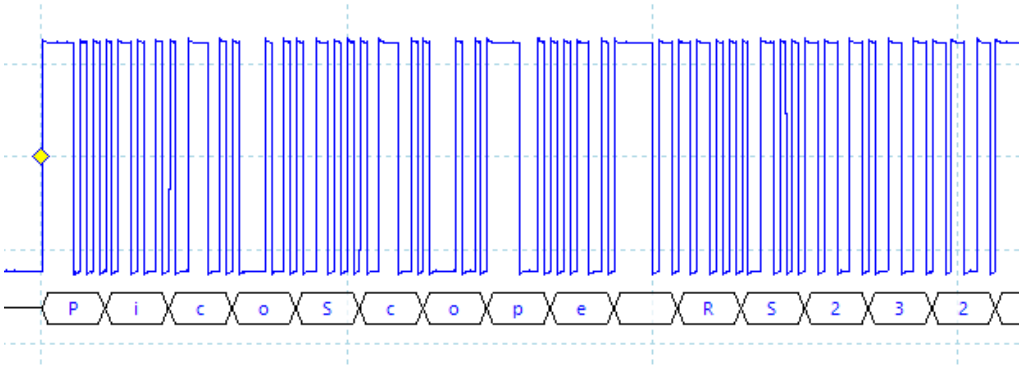
Typically, individual ASCII characters are transmitted as a sequence of 8 bits bounded by a start bit and a stop bit, with a bit order of LSB first and MSB last.

Electrically the voltage swing is relatively high and polarity is inverse, so that a Logic level 1 is a low voltage between –3 V and –15 V, and a logic level 0 is a high voltage between +3 V and +15 V.

RS-232 is a low-speed standard for data transfer, with a baud rate of 9600 being most commonly used. The low speed and short bursts of data have the advantage that the receiver is able to synchronize using the start bit alone and therefore no addition synchronization clock line is required.

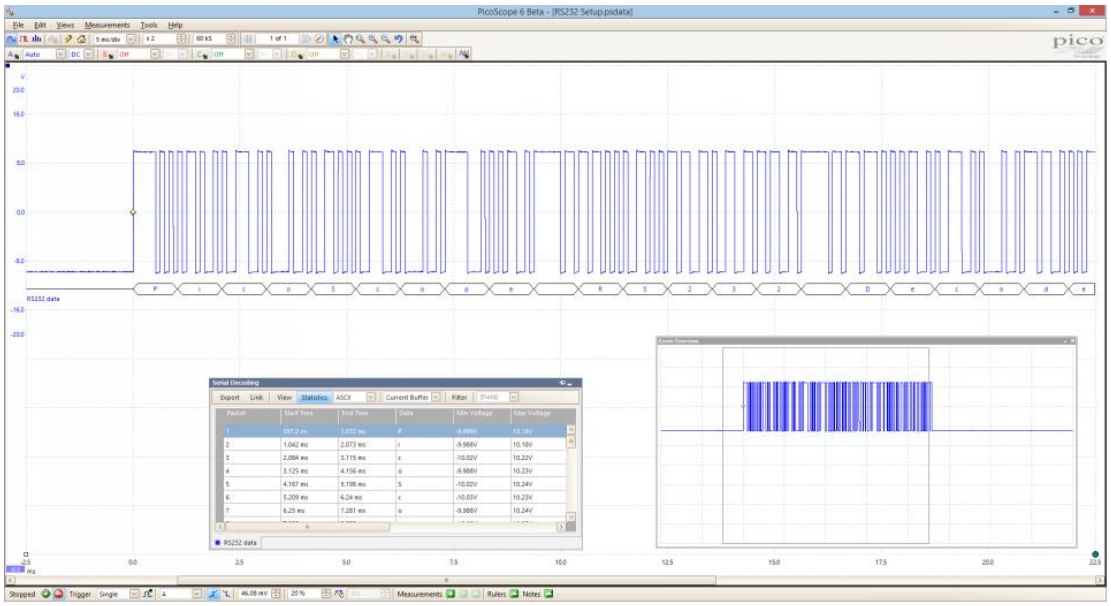
RS-232 decoding with PicoScope

RS-232 serial decoding is included in PicoScope as standard. The decoded data can be displayed in the format of your choice: **In Graph**, **In Table**, or both at once.



In Graph format shows decoded data in Binary, Hex, Decimal, or ASCII format, aligned with the analog waveform, on a common time axis. Decoded data can be zoomed and correlated with acquired analog channels to investigate timing errors or other signal integrity issues that are root cause of data errors.

In Table format shows a list of the decoded packets, showing data values with the packet start and stop times.



The PicoScope RS-232 decoder can also handle similar serial data standards such as RS-422 and RS-485

I2C

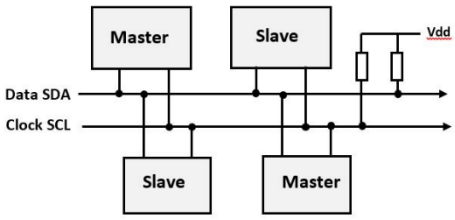
I2C (Inter Integrated Circuit) is a low-speed serial data protocol, commonly used to transfer data between multiple components and modules within a single device.

Developed in the early 1980s by Philips Semiconductors (now NXP), I2C employs 2 signal wires to transfer “packets” of information between one or more “master” devices such as microcontrollers, and multiple “slave” devices such as sensors, memory chips, ADC and DACs.

Wiring

Multiple “master” and “slave” I2C devices are connected to the bus using two lines:

- SCL – Serial Clock
- SDA – Serial Data



Signaling voltages are typically 0 V for logic low and +3.3 V or +5 V for logic high.

Pull-up resistors keep both lines at logic high level when the bus is idle.

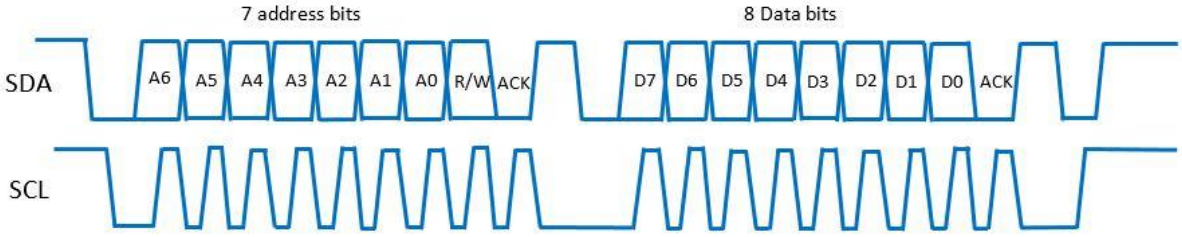
Signaling

I2C bus speeds range from 100 kbit/s in Standard mode, 400 kbit/s in Fast mode, 1 Mbit/s Fast mode plus, and 3.4 Mbit/s in High Speed mode.

Each device on the bus is recognised by a unique 7-bit or 10-bit address.

Data is transferred in “packets”, which include the address of the device, a read/write command, acknowledgements and the data being transferred.

The diagram shows the structure of a single packet of I2C data.



At the start of packet a master device takes control of the bus by driving SDA low while SCL remains high. This indicates that a message will follow.

Next a 7 (or 10) bit address is transmitted followed by a R/W bit to indicate whether it is a read (1) or write (0) instruction.

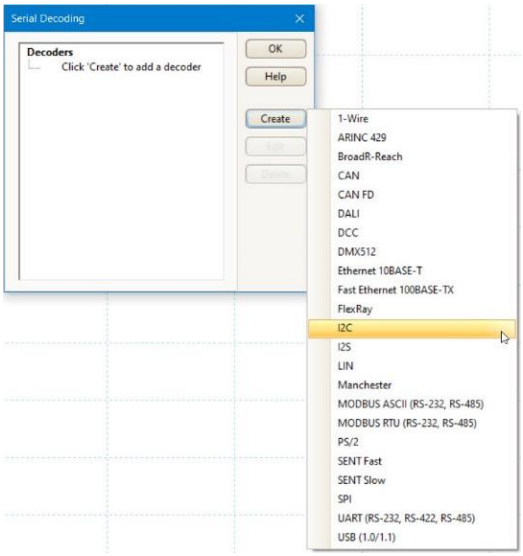
The addressed slave device then transmits an acknowledge (ACK) bit by pulling the SDA line low. If the line remains high, the master can infer that the slave did not recognise the address and corrective action needs to be taken.

After the address is acknowledged by the slave, the master continues to generate the clock and depending on the R/W bit either the master or the slave will send data over the bus. After each byte of data sent, an ACK is generated by the receiving device.

The end of packet is recognized by the SDA line going from low to high when the SCL is already high.

I2C decoding in PicoScope

I2C serial decoding is included in PicoScope as standard. Select I2C from the list of protocols in the **Tools** > **Serial Decoding** > **Create** drop-down menu.

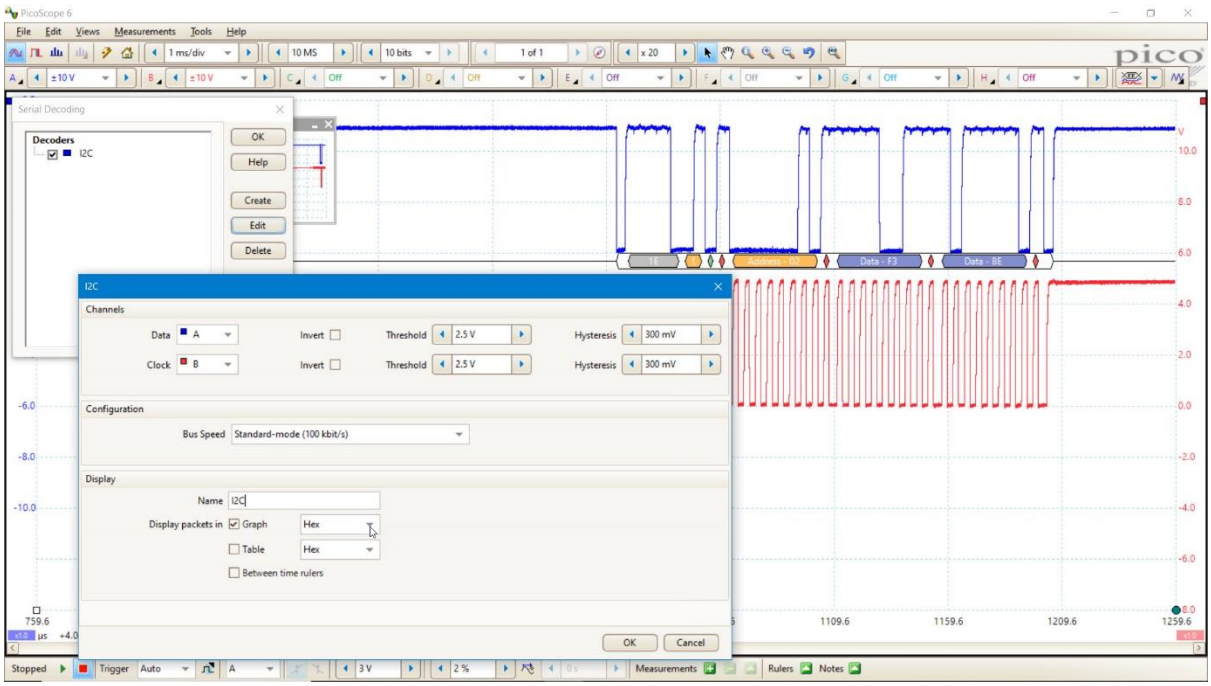


Settings

In the Settings box select the correct channels and thresholds for Data and Clock and choose the Bus speed to match the device under test.

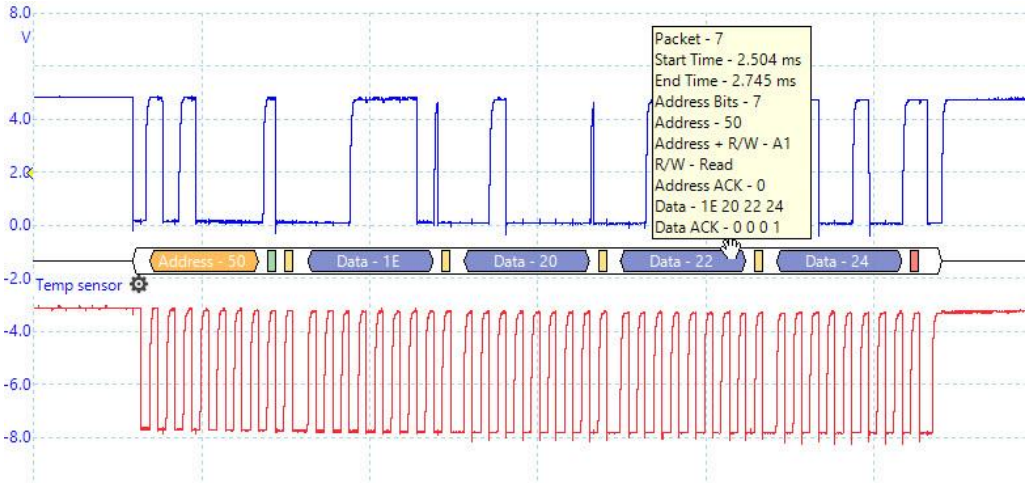
The bus can be given a name, such as "Temperature sensors", to make it easily readable.

Decoded data can be displayed in the format of your choice: Graph, Table, or both at once.



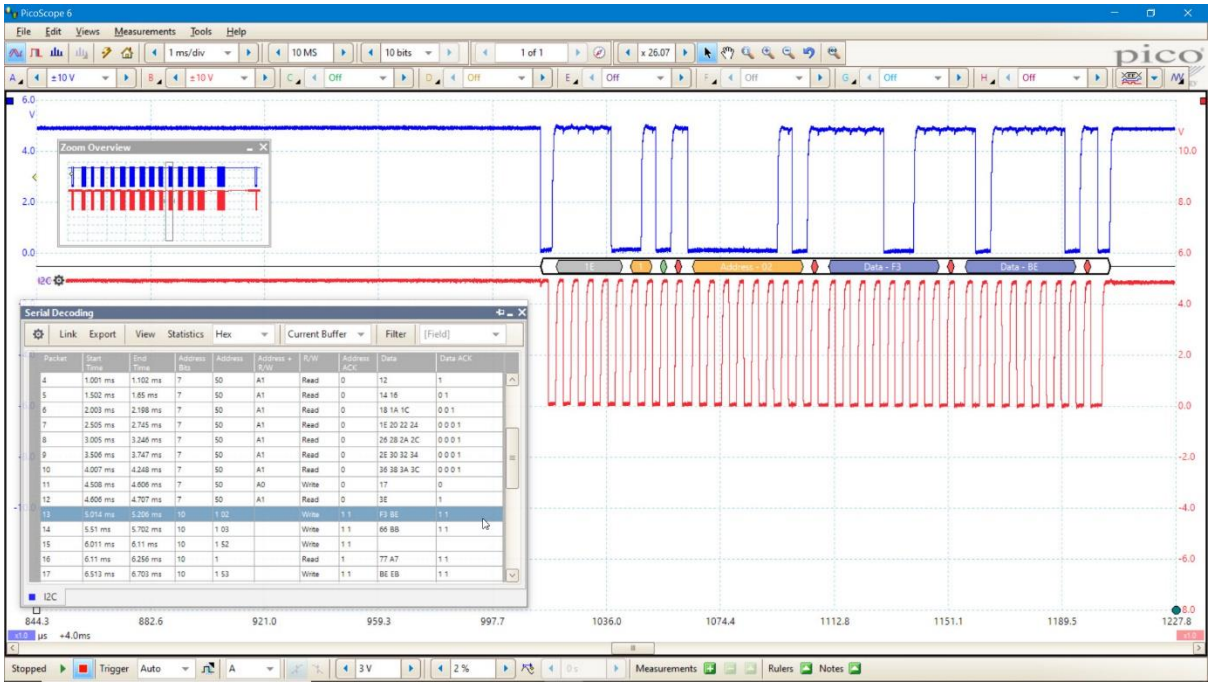
Graph

Shows decoded data in a bus format, aligned with the analog waveform, on a common time axis. Frames can be zoomed and correlated with acquired analog channels to investigate timing errors or other signal integrity issues that are the cause of data errors.



Table

Shows a list of the decoded frames, including the data and all flags and identifiers. You can set up filtering conditions to display only the frames or data you are interested in, search for frames with specified properties, or use a Link File to translate frame ID and hexadecimal data into human-readable form.



SPI bus

Introduction

SPI (Serial Peripheral Interface) bus was originally developed by Motorola for use with their microcontrollers. Due to the simplicity of the bus, other manufacturers adopted it and it has become widely available in components used in embedded system designs. It is commonly used for chip-to-chip communications between a CPU and keyboard, display, ADCs and DACs, real-time clocks, EEPROM, SD and other memory devices.

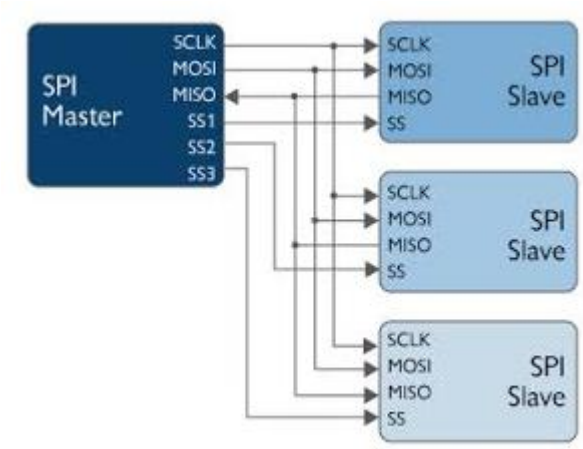
SPI is a synchronous bus with four lines: Data - master output/slave input (MOSI) and master input/slave output (MISO), clock (SCLK), and slave select (SS or CS). SPI is a full duplex standard, meaning signals can be transmitted in both directions simultaneously, with data rates from a few Mb/s to tens of Mb/s.

Devices communicate using a master-slave architecture with a single master. The master device initiates the frame for reading and writing. Multiple slave devices can be addressed with individual slave select lines.

Wiring

The SPI bus is a master/slave, 4-wire serial communications bus. The four signals are Data - master output/slave input (MOSI), master input/slave output (MISO), clock (SCLK), and slave select (SS or CS).

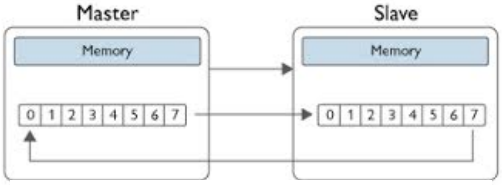
SPI connections, single slave device



SPI wiring master to three slave devices

Signaling

Whenever two devices communicate, one is referred to as the "master" and the other as the “slave”. The master drives the serial clock. Data is simultaneously transmitted and received, making it a full-duplex protocol. SPI uses the SS (or CS) line to specify which device data is being transferred to or from, so each unique device on the bus needs its own SS signal from the master. If there are 3 slave devices, there are 3 SS lines from the master, one to each slave.



SPI full duplex shift register

To begin communication, the bus master configures the clock, using a frequency supported by the slave device, typically a few MHz or tens of MHz. The master then selects the slave device with a logic level 0 on the select line.

During each SPI clock cycle, a full duplex data transmission occurs. The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it.

Transmissions involve two shift registers of a given word size, such as eight bits, one in the master and one in the slave. Data is usually shifted out with the most-significant bit first, while shifting a new least-significant bit into the same register. After the register has been fully shifted out the master and slave have exchanged register values. If more data needs to be exchanged the shift registers are reloaded and the process repeats. Transmission may continue for any number of clock cycles. When complete, the master stops toggling the clock signal, and deselects the slave.

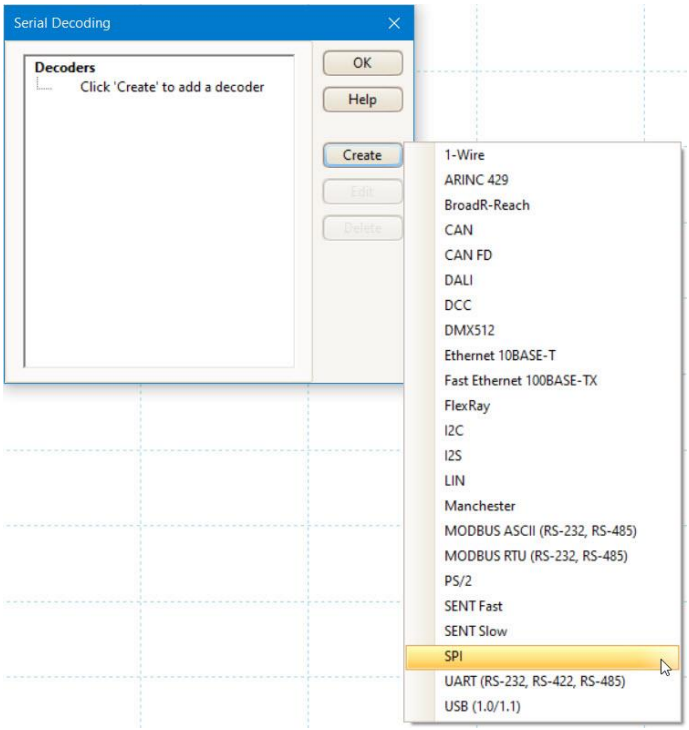
The master device must select only one slave at a time. Slave devices on the bus that have not been activated using their chip select line must disregard the input clock and MOSI signals, and must not drive MISO.

Capturing and analyzing SPI with PicoScope

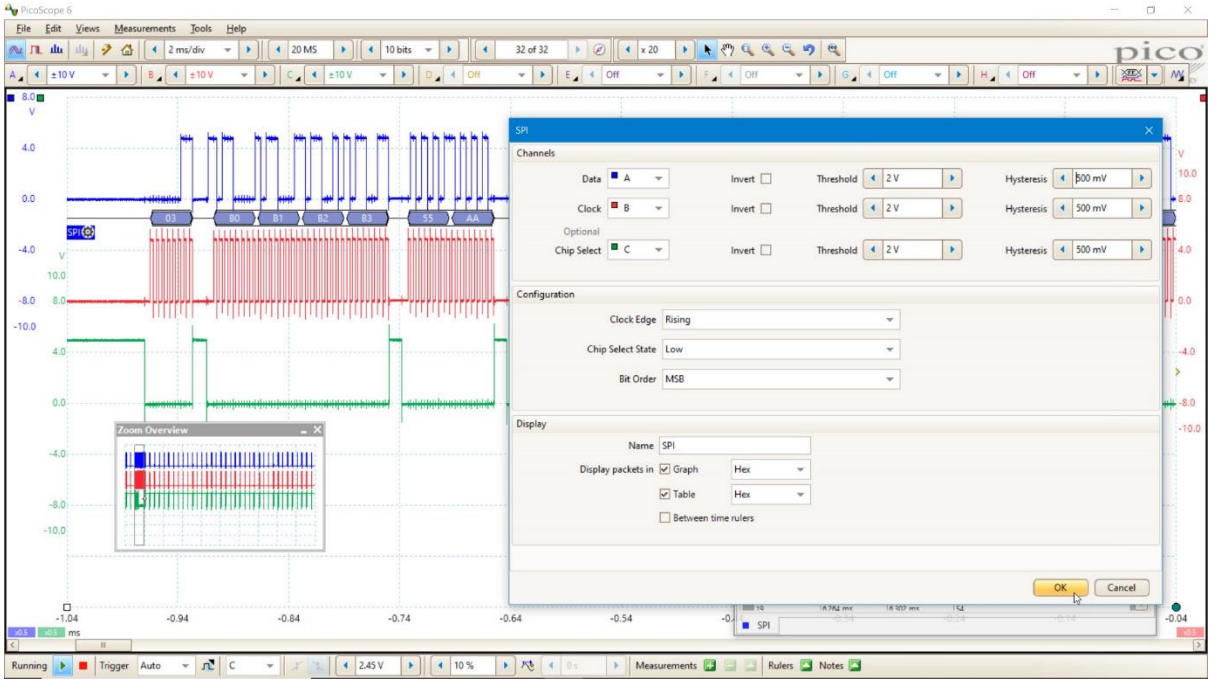
To decode SPI data first acquire the packets of interest using PicoScope.

Then select **Serial Decoding** from the **Tools** menu.

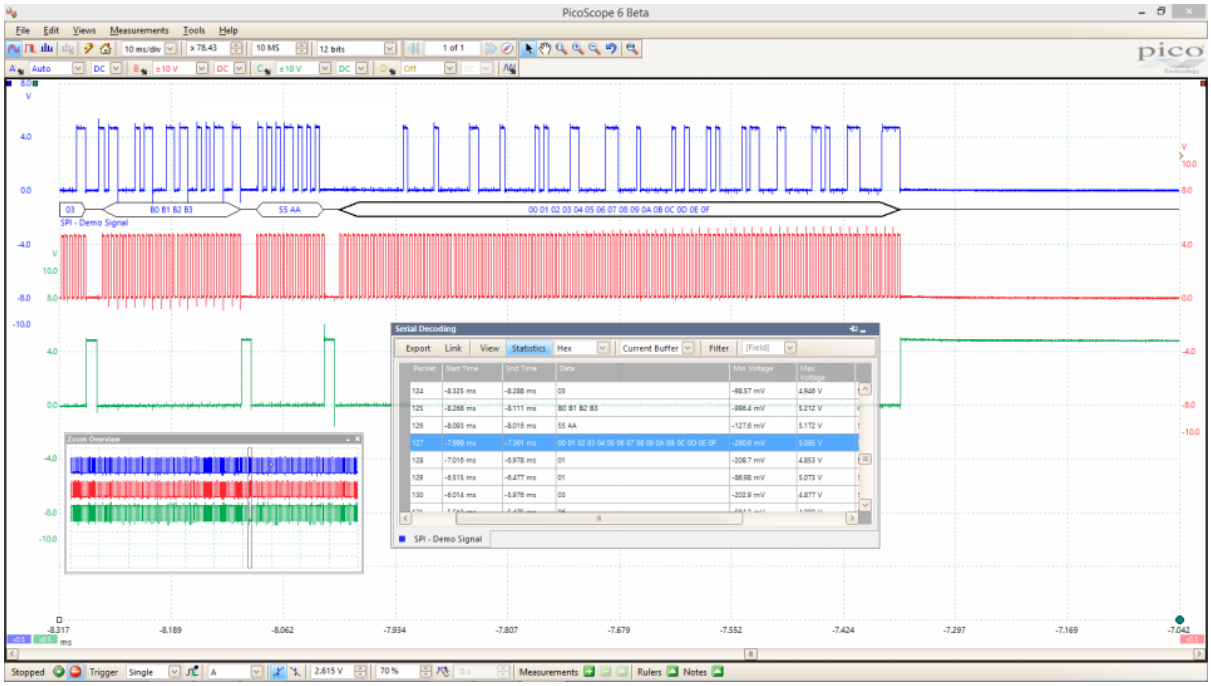
Click **Create** and select **SPI** from the list of available protocols.



Select the corresponding PicoScope input channels in the SPI configuration menu for Data (MOSI / MISO), Clock, Slave Select (SS). Set any other parameters as necessary. Click **OK** to see the decoded SPI packets in the PicoScope graph display.



In the SPI configuration menu, check both the In Graph and the **In Table** boxes to display SPI packets correlated in time with the acquired data channels plus a tabular listing of the packets. Double-click a packet in the graph view to see the same packet in the table view, and vice versa.



CAN and CAN FD bus decoding

CAN bus (Controller Area Network) is a serial data standard originally developed in the 1980s by Robert Bosch GmbH for use in automotive applications. Today it is also widely used in industrial process control and aerospace applications.

It allows microcontrollers and electronic devices to communicate with each other without using a host computer and provides fast and reliable data transfer in electrically noisy environments at low cost and with minimal wiring.

CAN employs differential signalling to provide a high level of immunity to electrical noise.

In 1991 Bosch published the specification for CAN 2.0, which details two formats:

- **CAN 2.0A** is the standard format with an 11-bit identifier.
- **CAN 2.0B** is the extended format with a 29-bit identifier.

In 1993 ISO (International Organization for Standardization) released the CAN standard ISO 11898, which was later restructured into three parts:

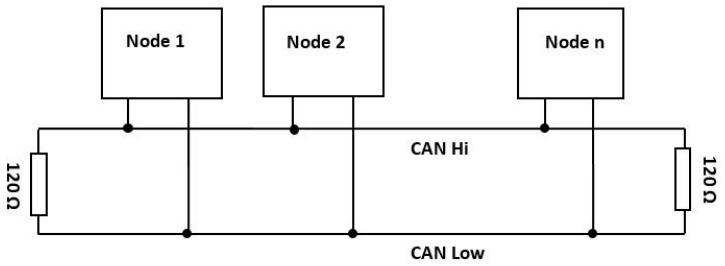
- **ISO 11898-1** which covers the data link layer
- **ISO 11898-2** which covers the CAN physical layer for high-speed CAN (up to 1 Mbit/s).
- **ISO 11898-3** which covers the CAN physical layer for low-speed, fault-tolerant CAN (up to 125 kbit/s)

Bosch subsequently released **CAN FD 1.0** or CAN with Flexible Data-rate, which was incorporated into **ISO 11898-1:2015**. This specification allows for increased data lengths as well as optionally switching to a faster bit rate after the arbitration is decided. CAN FD is reverse-compatible with existing CAN 2.0 networks, so new CAN FD devices can coexist on the same network with existing CAN devices.

Wiring

Many devices can be connected to the CAN bus, ranging from complex electronic control units to simple I/O devices. Each device is called a node.

Each CAN node transmits differentially over two wires: CAN High and CAN Low.



Signaling

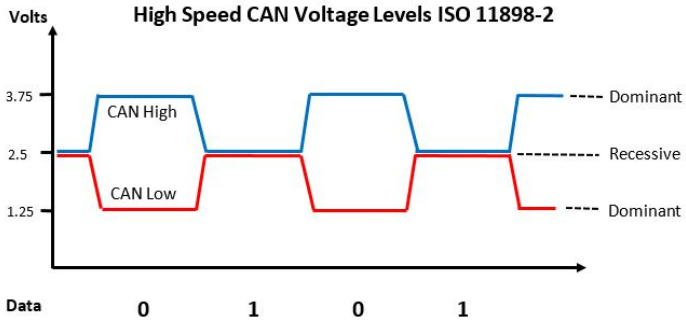
CAN transmits differentially over two lines, CAN High and CAN Low. There are two logic states:

- Dominant – Logic 0
- Recessive – Logic 1

Voltage levels vary according to the spec. Here we look at High Speed CAN.

During a recessive (logic 1) transmission the bus is not actively driven and rests at around 2.5 V.

During a dominant (logic 0), CAN High is driven towards 5 V (or Vcc) and CAN Low is driven towards 0 V.



Arbitration

CAN data is sent in Frames starting with a dominant 0 followed by an Identifier, which forms the basis of arbitration (Priority) where two or more nodes attempt to transmit at the same time.

Each node is assigned an Identifier which can be 11 bits (CAN 2.0A) or 29 bits (CAN 2.0B) in length.

The table shows three nodes attempting to transmit at the same time, each starting with dominant 0s. When a node transmits a recessive 1 but sees that the bus remains at dominant 0, it realizes there is a contention and ceases to transmit and waits for the next opportunity to transmit.

In this way the node with the lowest value ID wins arbitration and is given priority to transmit the rest of the frame.

	Start	ID Bits											Rest of Frame					
	Bit	10	9	8	7	6	5	4	3	2	1	0						
Node 2	0	0	0	1	Stops transmitting													
Node 5	0	0	0	0	0	0	1	0	0	1	1	0	X	X	X	X	X	X
Node 14	0	0	0	0	0	0	1	1	Stops transmitting									

Frames

CAN has four frame types:

- **Data frame**: a frame containing node data for transmission.
- **Remote frame**: a frame requesting the transmission of a specific identifier.
- **Error frame**: a frame transmitted by any node detecting an error.
- **Overload frame**: a frame to inject a delay between data and/or remote frame.

A CAN network can be configured to work with two different frame formats: the base frame format (CAN 2.0A & CAN 2.0B) which supports 11-bit identifiers, and the extended frame format (CAN2.0B only) which supports 29-bit identifiers by allowing the addition of an 18-bit identifier extension.

The table below shows the format for a CAN Data Frame with Base Format (11-bit) with no bit stuffing

An **identifier extension bit (IDE)** determines if the 18-bit ID extension is being used.

The RTR bit (Remote transmission request) determines between data frames (0) and remote frames (1)

The Data Length Code indicates the length of data in bytes; in this case 1 byte.

CRC stands for cyclic redundancy check, which is used for error detection.

Ack slot bit – All nodes that receive a frame without finding any errors transmit a dominant 0, which overrides a recessive 1 sent by the transmitter. If the transmitter detects a recessive 1, it knows that the frame was not received correctly.

End of frame is confirmed by the transmission of 7 recessive 1s.

Complete CAN Data Frame

		Arbitration Field – 11 bit identifier											Control			Data length code				Data – 1 to 8 bytes								CRC Field (15 bits) – 15 Bits																	End of Frame – 7 bits						
	Start of Frame	ID 10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR	IDE	Reserved	DL 3	DL 2	DL 1	DL 0	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	CRC14	CRC12	CRC11	CRC10	CRC9	CRC8	CRC7	CRC6	CRC5	CRC4	CRC3	CRC2	CRC1	CRC0	CRC Delimiter	Ack slot bit	Ack delimitter	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Data	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1

Complete CAN data frame structure

Bit-stuffing

To ensure enough transitions to maintain synchronization, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. Bit-stuffing does not occur during the CRC delimiter, ACK field and end of frame.

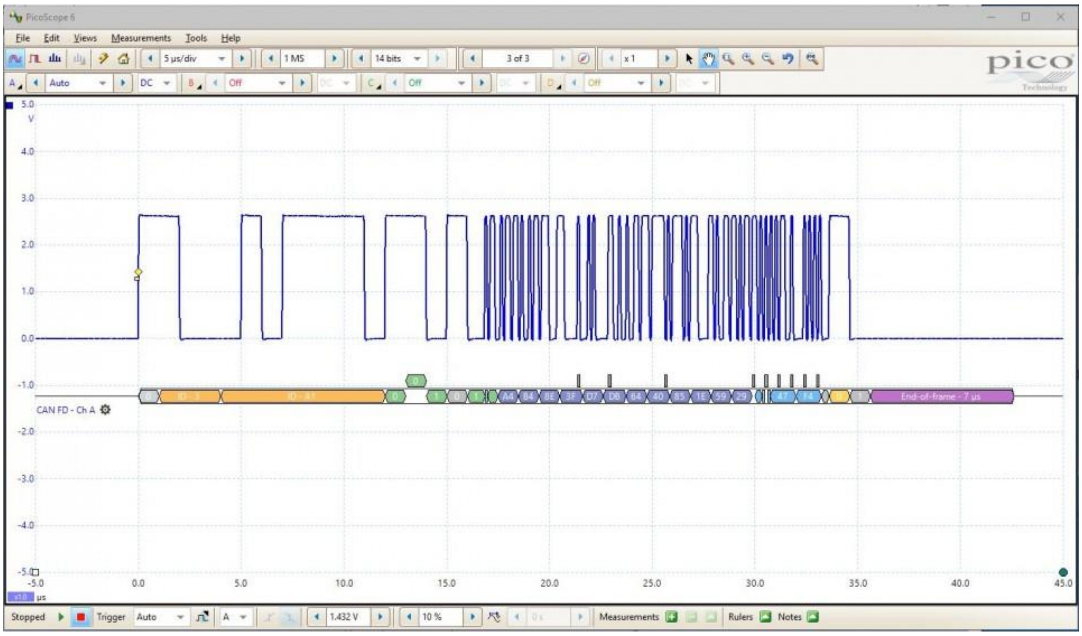
In the fields where bit-stuffing is used, six consecutive bits of the same type (000000 or 111111) are considered an error and an active error flag consisting of six consecutive dominant bits can be transmitted by a node when an error has been detected.

CAN FD

Bosch followed the CAN standard with CAN FD 1.0 or CAN with Flexible Data-Rate, which later became part of the ISO 11898-1:2015 standard. This specification allows for increased data lengths as well as optionally switching to a faster bit rate after the arbitration is decided.

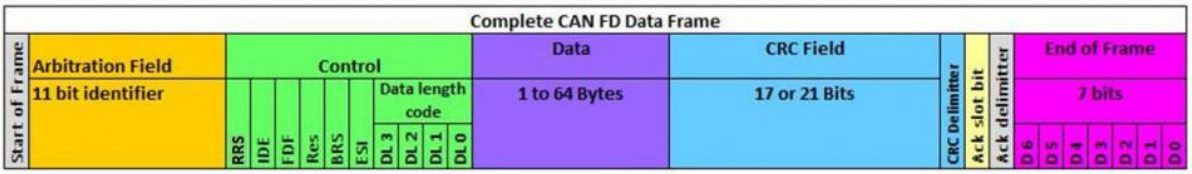
CAN FD meets the growing need to transfer more data, more quickly, in automotive (and other) systems of increasing complexity.

CAN FD is reverse-compatible with existing CAN 2.0 networks so new CAN FD devices can coexist on the same network with classical CAN devices.



Differences in protocol between CAN and CAN FD

The table shows the structure of a CAN FD frame.



Arbitration and the use of Base and Extended frame formats are identical in both classical CAN and CAN FD.

As CAN FD does not use remote frames, the RTR bit used in CAN is redundant and is replaced with the Remote Request Substitution bit (RRS), which is always transmitted dominant (0).

The IDE bit is used in the same way.

The reserve bit in CAN now becomes the FDF bit (Flexible Data Format) and is a dominant 0 to indicate that the frame is in classic CAN format. Frames sent in CAN FD format are indicated with a recessive 1.

Next is a reserve bit for future use.

Then comes the BRS bit (Bit Rate Switch). If BRS is sent dominant, the bit rate remains the same across the whole frame. If BRS is recessive, a higher bit rate will be transmitted after this bit up to and including the CRC delimiter.

The ESI bit (Error State Indicator) is a dominant transmission for error active, and recessive transmission for error passive.

Data Length Code

The number of bytes in the Data Field is indicated by the Data Length Code (DLC).

Its coding is different in CAN and in CAN FD.

The first nine codes are the same, but the following codes, that in CAN all specify a DATA FIELD of eight bytes, specify longer DATA FIELDS in CAN FD.

	No of Bytes	Data Length Code			
		DL3	DL2	DL1	DL0
Codes in CAN and CAN FD Format	0	0	0	0	0
	1	0	0	0	1
	2	0	0	1	0
	3	0	0	1	1
	4	0	1	0	0
	5	0	1	0	1
	6	0	1	1	0
	7	0	1	1	1
CAN format	8	1	0/1	0/1	0/1
Codes in CAN FD Format	8	1	0	0	0
	12	1	0	0	1
	16	1	0	1	0
	20	1	0	1	1
	14	1	1	0	0
	32	1	1	0	1
	48	1	1	1	0
	64	1	1	1	1

CAN FD CRC field

Because of the longer data lengths with CAN FD, more bits are required for the CRC check. If the frame holds 16 bytes or less, a CRC with 17 bits is used; and if the CAN frame holds more than 16 bytes, a CRC with 21 bits is used.

After the CRC field the Ack and end of frame structure is the same as with classical CAN.

CAN bus debugging

Errors can occur due to inductors, coils and power devices which can cause large voltage spikes, noise and ringing. An increasing number of embedded computers and devices are being added to automobile CAN buses and as more nodes are added the available bus time becomes more occupied. When the traffic reaches around 40% of the bus time, errors can start to occur. At this point an oscilloscope may be required to debug the network.

To monitor and find faults on a CAN bus it is important to have an oscilloscope with deep memory to capture a large time window with multiple frames of data. The instrument can then process the entire acquired waveform and then zoom in to analyze the data packets.

We recommend that the instrument has a bandwidth of ten times the CAN baud rate, to analyze rise times and any fault conditions.

Step one – probing

CAN is a differential signal, CAN Low being the inverse of CAN High. Viewing the difference between the two removes any common-mode interference encountered by the signal during transmission.

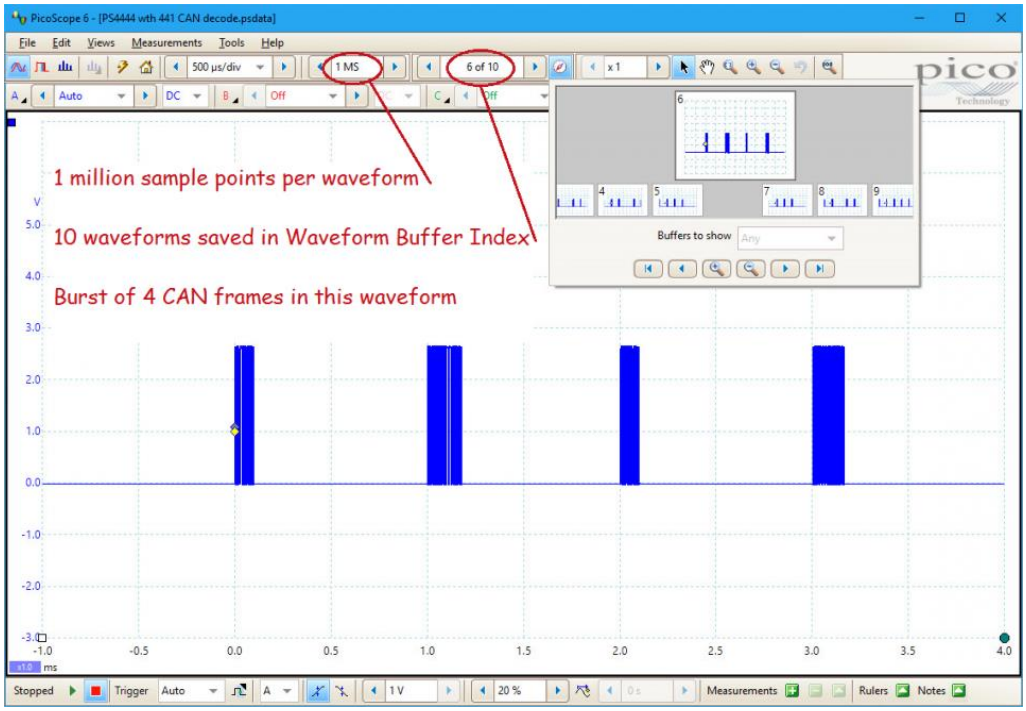
Best results will be achieved by acquiring the signal difference between CAN Low and CAN High using a differential probe or a differential input oscilloscope such as the PicoScope 4444.

The signal can still be acquired using a single-ended probe connected to either CAN Low or CAN high, but any common-mode noise will be displayed and may cause errors in decoding on the oscilloscope which would not affect the CAN receiver.

Step two - acquire the CAN data signal

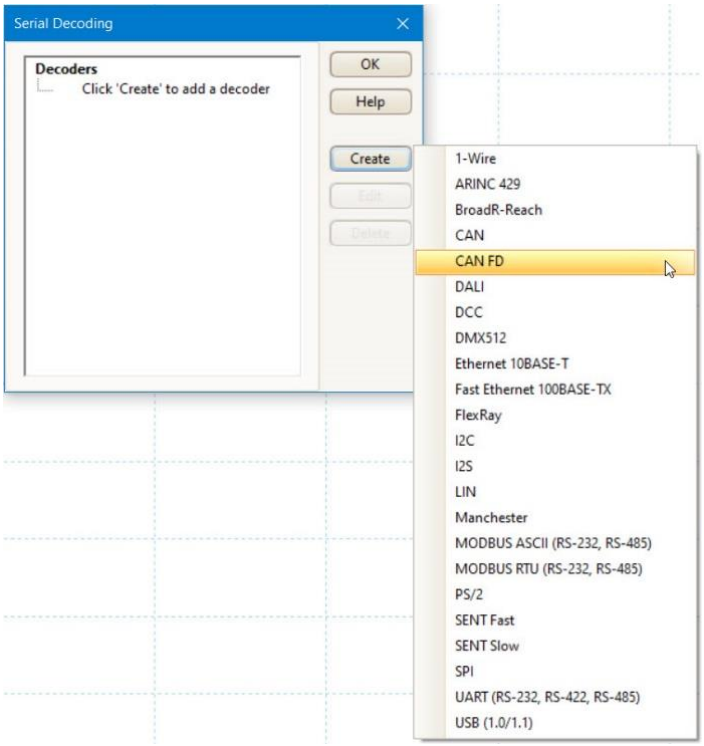
Set memory length to enough to acquire as many frames as required, and with the resolution required to resolve individual bits.

Or use the buffer memory index to capture short bursts of frames while ignoring any dead time in between.



Step three – set up a decoder

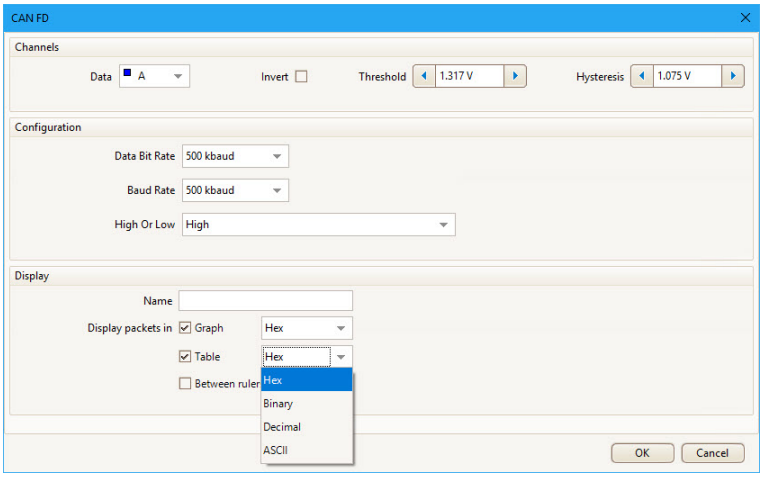
Select Serial Decoding from the **Tools** menu, click **Create**, then select **CAN** or **CAN FD**.



Select **Channel > Data > A** and **Configuration** either **Low** or **High** if probing single ended, or **High** if probing differentially.

PicoScope will automatically calculate the optimum threshold, hysteresis and baud rate but these values can be customized if necessary.

In the **Display** menu tick **Graph** or **Table**, or both, and then select the required format—**Hex/Binary/Decimal/ASCII**—and click **OK**.



If **Graph** is selected, a color-coded trace will appear in the graph display, time-correlated with the acquired data.

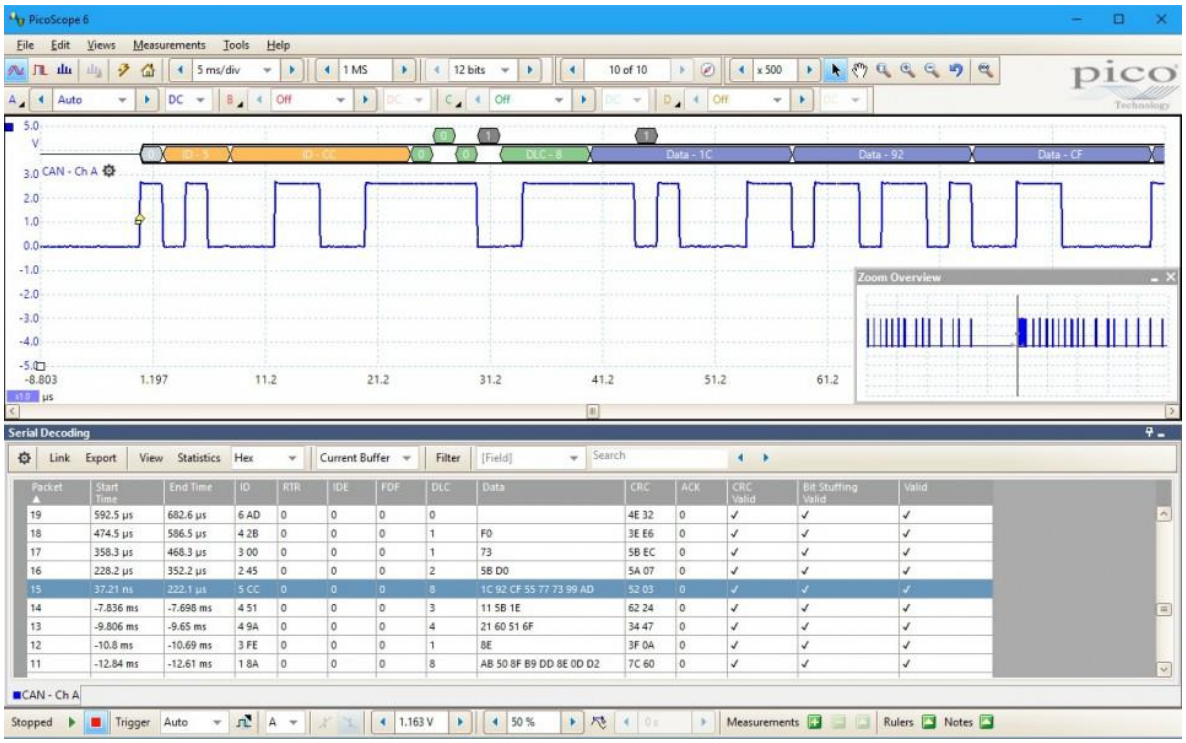
If **Table** is selected, all the data will be presented in a table format. There are several useful features available with the table:

Double-click a frame in the graph format and the corresponding frame will be highlighted in the table.

Select **Export** to save the table data in .csv format.

Set up a Link file so that data in the table can be displayed as meaningful text.

Filter on the table to search any field for specific values, for example invalid CRCs.



Summary

CAN and CAN FD decode is a standard feature in the PicoScope software and can be used with all real-time PicoScope oscilloscopes.

ARINC 429

Introduction

ARINC 429, also known as the Mark 33 DITS specification, was developed to provide interchangeability and interoperability of line replaceable units (LRUs) in commercial aircraft. ARINC 429 defines the physical and electrical interfaces of a two-wire data bus and the data protocol to support an aircraft's avionics local area network.

The physical connection wires are twisted pairs carrying balanced differential signalling. Data words are 32 bits in length and most messages consist of a single data word. Messages are transmitted at either 12.5 or 100 Kbit/s. The transmitter constantly transmits either 32-bit data words or the NULL state. Most ARINC messages contain only one data word consisting of either binary (BNR), binary coded decimal (BCD), or alphanumeric data encoded using ISO Alphabet No. 5. File data transfers that send more than one word are also allowed.

ARINC 429 employs several techniques to minimize electromagnetic interference (EMI) with on-board radios and other equipment.

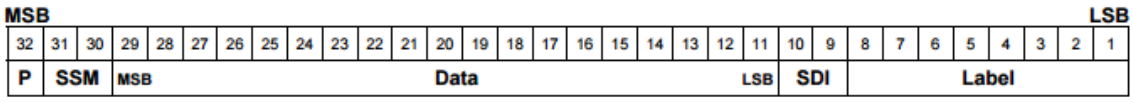
In addition to twisted-pair cabling, ARINC signalling defines a 10 V peak differential signal, with acceptable voltage rise and fall times for the Data A and Data B levels, and complementary differential bipolar return-to-zero (BPRZ) data encoding to minimize EMI emissions from the cable itself.

ARINC 429 word format

ARINC 429 data words are made up of five primary fields:

- Parity – 1 bit
- Sign/Status Matrix (SSM) – 2 bits
- Data – 19 bits
- Source/Destination Identifier (SDI) – 2 bits
- Label – 8 bits

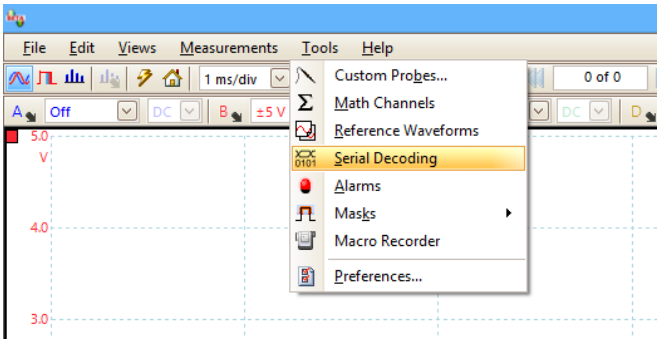
Although ARINC 429 word transmission begins with Bit 1 and ends with Bit 32, ARINC 429 words are typically shown in the order from Bit 32 to Bit 1.



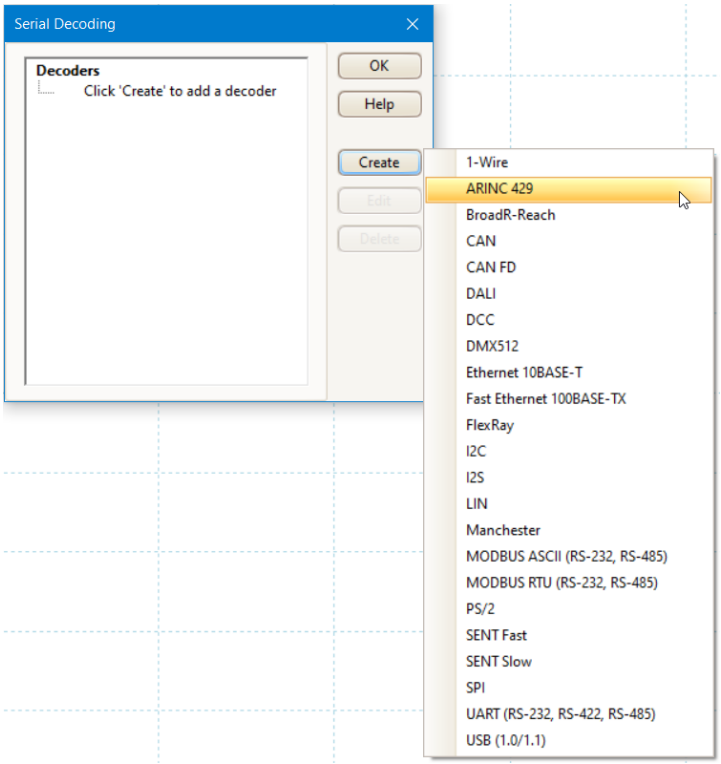
ARINC 429 32 bit Word Format

Decoding ARINC 429

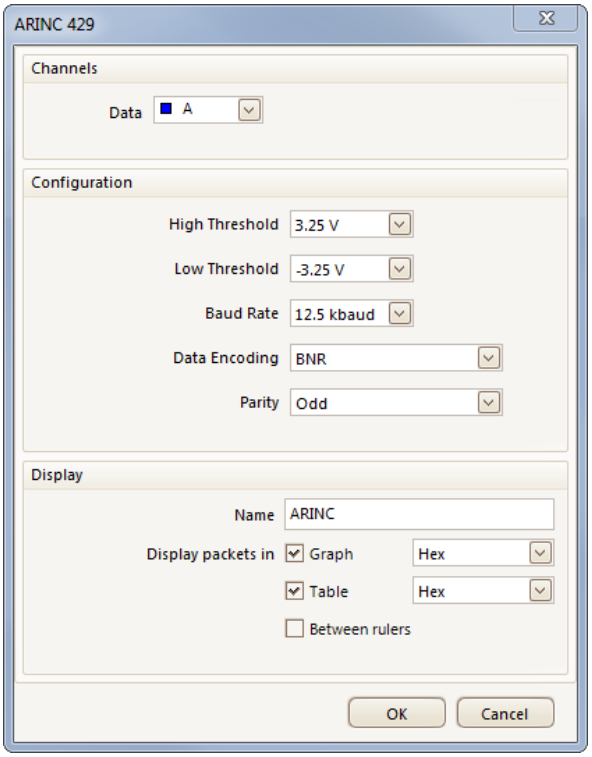
The first step is to acquire the ARINC word of interest using PicoScope. Then select **Serial Decoding** from the **Tools** menu.



Click **Create** and select **ARINC 429** from the list of available protocols.

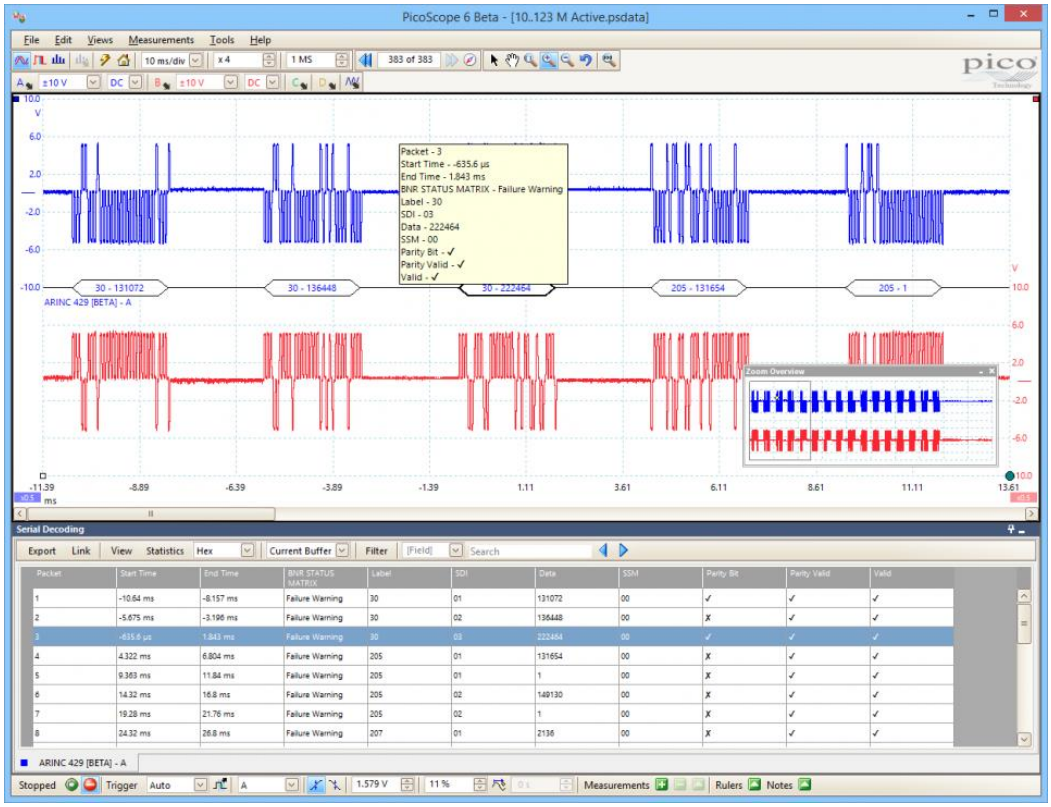


In the ARINC 429 configuration menu select the appropriate PicoScope input channel, baud rate and any other parameters as necessary. Click **OK** to see the decoded ARINC 429 messages in the PicoScope graph display.



PicoScope displays the ARINC 429 decoded packet data correlated with the captured waveform.

Check the **In Table** box to add a tabular listing of the ARINC 429 packets.



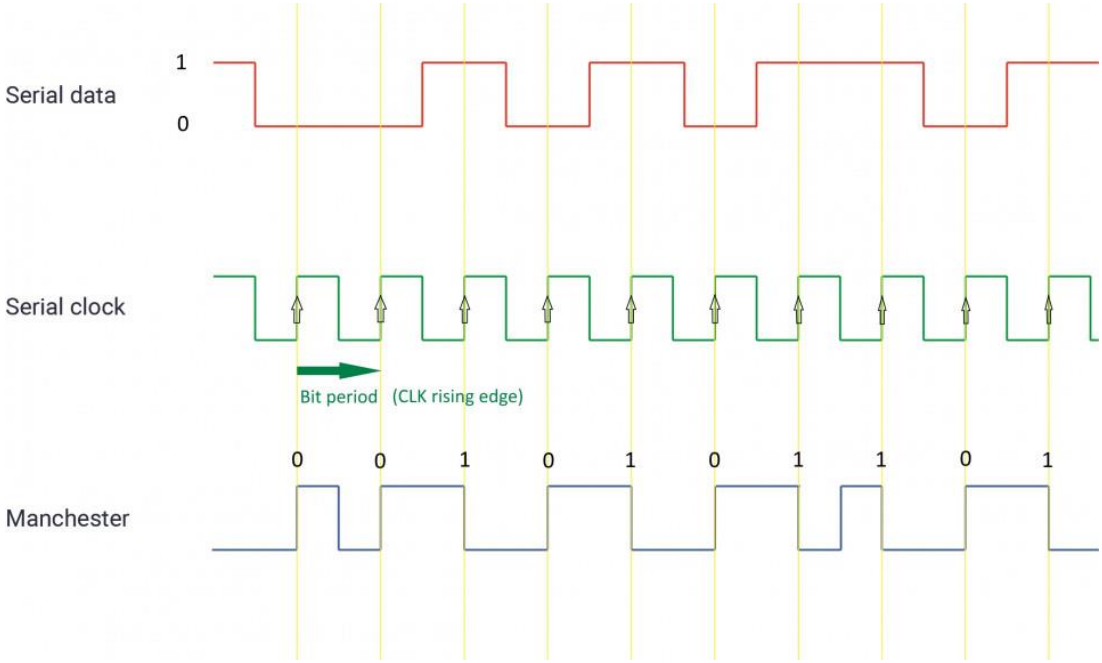
Manchester Encoding

Introduction

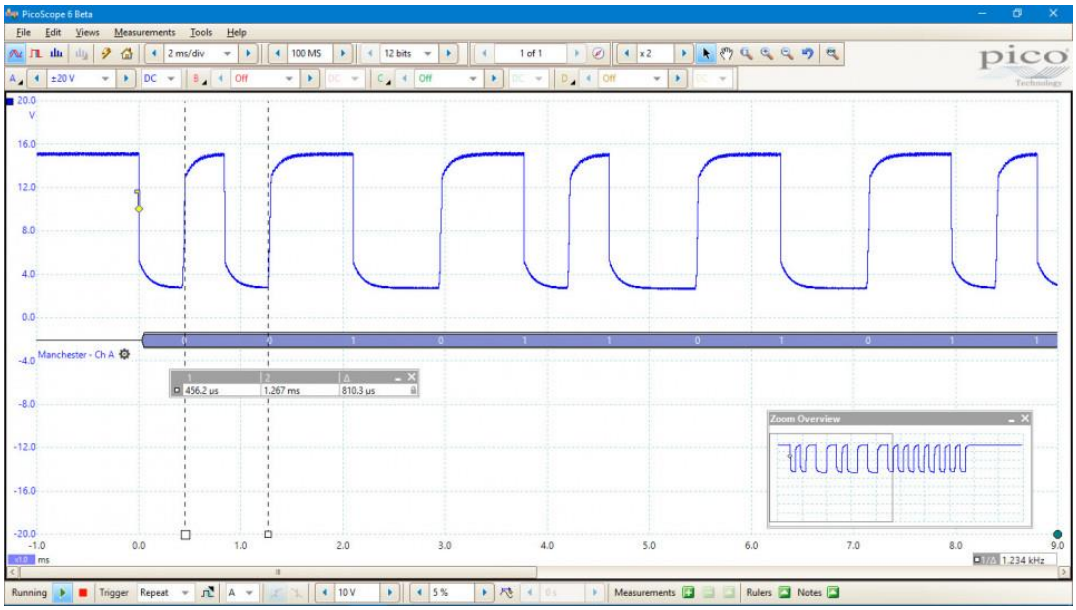
Manchester encoding is a serial data signaling system originally developed at the University of Manchester for use on early generation computer systems with magnetic drum and 1600 bpi magnetic tape data storage devices. It is in widespread use today in network communications such as 10BaseT Ethernet and MIL-STD-1553, as well as consumer IR devices and DALI lighting controls.

Manchester waveforms are “self-clocked”; the clock signal is embedded in the transmitted data using an exclusive-OR Boolean function. Embedding the clock means that only one signal line is needed, rather than two or three that are required with standards such as I2C and SPI. This makes for simpler and lower-cost network wiring layout, and has several other advantages:

- Removes the need to manually preset and match transmitter and receiver baud rates.
- Clock / data skew is eliminated. Receiving devices are tolerant of jitter and frequency drift from the transmitter.
- In some cases, such as wireless and optical transmission networks, there isn’t a simple or low-cost way to add a second channel for the clock, which mandates a self-clocking technique.
- The embedded clock guarantees one or two edge transitions each bit period, which means that the transmitter and receiver can be AC coupled and therefore galvanically isolated from the network wiring. This avoids common-mode overvoltage problems and gives protection if a short circuit occurs in the network.
- Guaranteed transitions each bit period, even in the event of long sequences of 1s or 0s, enables straightforward recovery of the clock and data signal by the receiver.



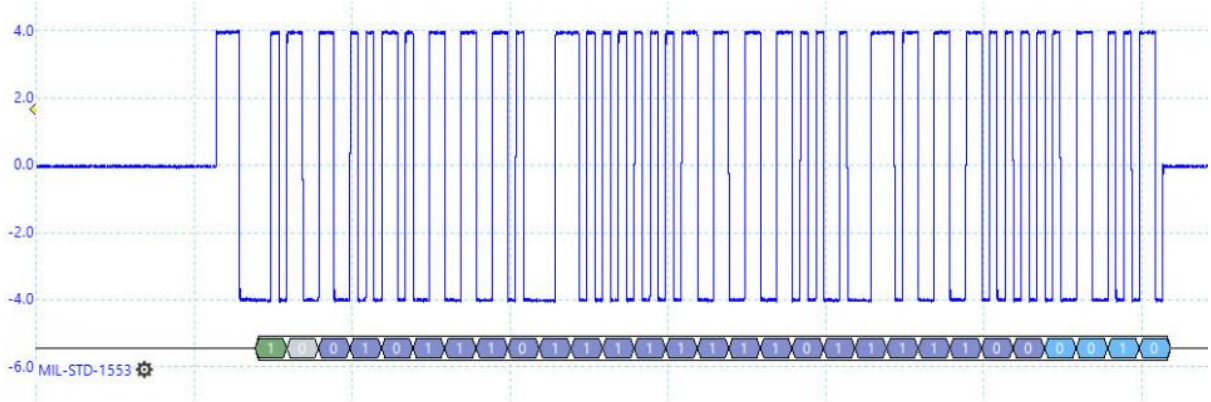
As with any engineering design choice, these advantages don't come for free. Embedding the clock doubles the required network bandwidth. But for many applications Manchester is a good and widely used communications protocol.



Manchester encoding uses an exclusive-OR Boolean function to combine the clock and data on a single channel. This means that binary information is conveyed as transitions rather than levels.

The transmitted Manchester data is clocked on the rising edge of the serial clock.

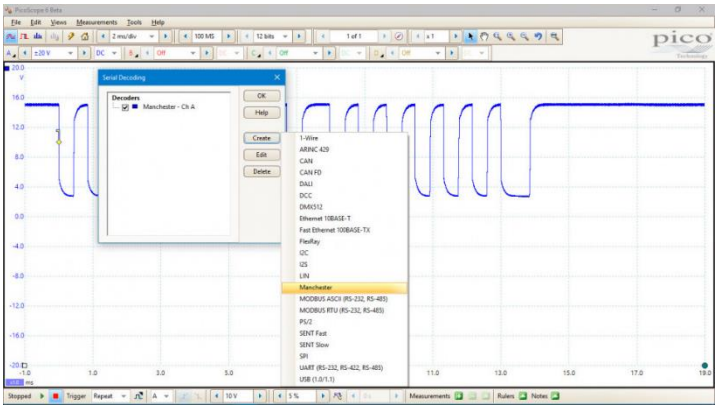
Note that there is always at least one transition per bit period, to enable clock recovery by the receiver and AC coupling across the network. The receiving device has to detect the timing of the fastest complete cycle of the incoming waveform to work out the bit period. From there, low-to-high transitions are received as binary zeros, high-to-low transitions as ones. (Or vice versa – depending on how the standard is implemented.)



MIL-STD-1553 Manchester encoded waveform

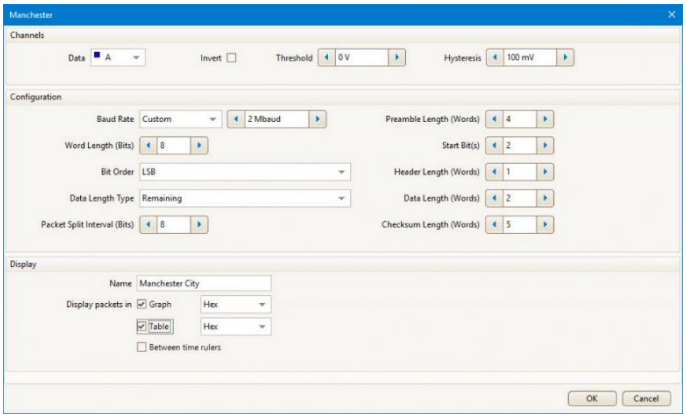
In the absence of any data to be sent, or when there are long sequences of zeros or ones, the clock information is still transmitted to keep the receiver synchronized.

Manchester waveforms can be decoded with PicoScope. From the **Tools** menu select **Serial Decoding**, then **Create**, and select **Manchester** from the list of available protocols.



Transmitted Manchester data is normally sent in packets that are arranged in fields that make up the complete protocol per the design specification for a particular application. Field settings include:

- Word length. Typically 8 to 64 bits
- Bit order: LSB or MSB first
- Start bits
- Preamble length (words)
- Header length (words)
- Data length (words)
- Checksum length (words)

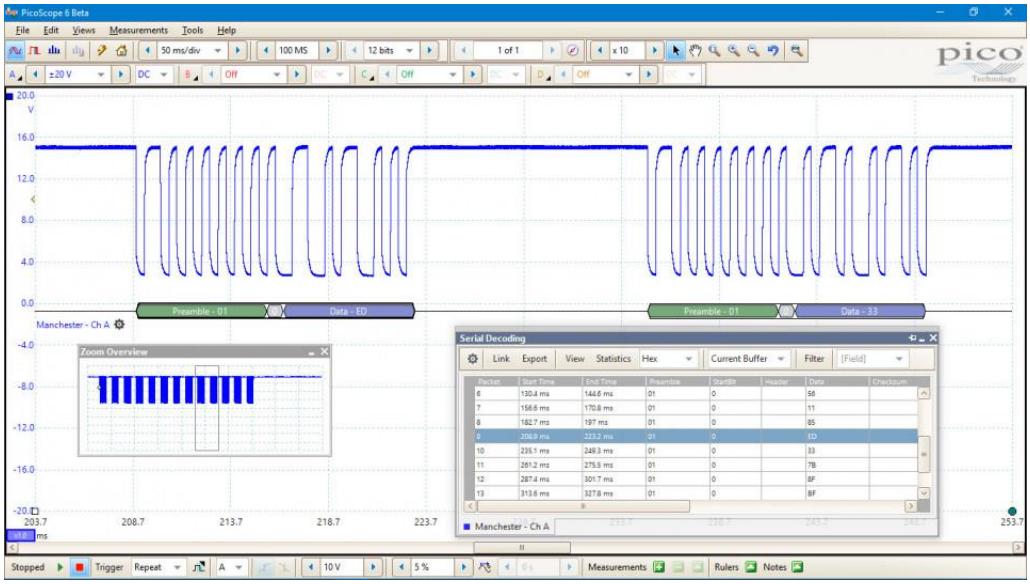


The PicoScope Manchester decoder settings panel allows each field to be set up to match the system design.

Decoded waveform data can be displayed:

- In graph: aligned with the captured waveform
- In table: listed in a tabular format

Double-clicking on a line in the table highlights the corresponding packet in the graph display.



More information

PicoScope Software is free to download from the Pico Technology website at https://www.picotech.com/downloads

It works with all Pico real time PC oscilloscopes and includes many more serial protocol decoders than covered here.

More protocol articles can be found at https://www.picotech.com/library/a-to-z, including:

- BroadR-Reach
- DALI
- Digital Command Control (DCC)
- FlexRay
- Modbus
- 1-Wire
- SENT bus
- USB

Pico Technology Ltd
James House
Colmworth Business Park
St Neots, Cambridgeshire
PE19 8YP
United Kingdom

www.picotech.com
www.picoauto.com
+44 (0) 1480 396 395