Debugging Serial Buses in Embedded System Designs

With Tektronix Oscilloscopes
Agenda

- Introduction

- Serial Data Buses
  - I²C
  - SPI
  - Audio
  - RS-232
  - CAN
  - LIN
  - FlexRay

- Troubleshooting Your Device
  - 2Decode of Serial Data
  - Capture and Search for Specific Messages
  - Characterize System Timing
  - Trace Data Flow Through a Network
  - Verifying System Behavior
  - In-Depth Analysis of Network Performance

- Summary
Transition from Parallel to Serial Buses

- Traditional way to connect digital devices used parallel buses

- Advantages
  - Simple point-to-point connections
  - All signals are transmitted in parallel

- Disadvantages
  - Occupies a lot of circuit board space
  - All connections must be the same length
  - Many connections limit reliability
  - Connectors may be very large

- With serial buses, these disadvantages are minimized
Design Implications of Serial Communication

- **Product Design**
  - Integrated into many processors, ASICs, and FPGAs
  - Easier signal routing
  - Less space, less weight, less power
  - Higher manufacturing yields and reliability
  - *Improves circuit board designs, lowers cost and reduces form factor*

- **System Design**
  - SPI and I²C buses enable connection of many ICs without external components.
  - CAN enables connection of sub-assemblies in automotive applications, and industrial controls in factory automation applications.
  - *Simplifies designs of complex systems.*
Debug Challenges of Serial Data

- **Parallel Buses**
  - Each line has its own signal path
  - Clock is generally a separate line
  - Easy to decode
  - State and Pattern triggering and decoding are straightforward with a logic analyzer or mixed signal oscilloscope

- **Serial Buses**
  - Signals are spread over time
  - Clock is sometimes embedded
  - Decode is tedious
  - Must decode first to trigger on packet
  - Analysis solutions available on some oscilloscopes

Serial data complicates bus troubleshooting
Serial Bus Review

- **Separate clocks:**
  - I²C
  - SPI
  - Audio (I²S, LJ, RJ, TDM)

- **Embedded clocks:**
  - RS-232 (RS-422, RS-485, UART)
  - CAN
  - LIN
  - FlexRay
I²C (Inter-Integrated Circuit)

- Used for chip-to-chip communication between microcontrollers and A/Ds, D/As, FPGAs, sensors, etc.
- Uses two single-ended, bi-directional signals: clock and data
- Any I²C device can be attached to the bus
- Data rates:
  - Standard Mode (100 kbps)
  - Fast Mode (400 kbps)
  - High Speed Mode (3.4 Mbps)
SPI (System Peripheral Interface)

- Used primarily to communicate between microcontrollers and their immediate peripheral devices
- Typical configuration has four signals: SCLK, MOSI, MISO, SS
  - Data is simultaneously transmitted and received
  - SS line used to specify slave device
  - Each unique device on bus has its own SS signal from master
- Multiple bus configurations are allowed
  - Network can use 2-, 3-, or 4-wire bus topology
- Data rates up to 10 Mbps

- SS – enables slave device to accept data
- MOSI – data from the master to a slave
- MISO – data from a slave to the master
- SCLK – serial clock driven by Master

![Diagram of SPI signals](image)
Audio

- Used for communicating serial audio data between ICs in audio electronic products. Examples include:
  - Inter-IC Sound (I²S)
  - Left Justified (LJ)
  - Right Justified (RJ)
  - Time Division Multiplexed (TDM)

- Synchronous, bi-directional serial interfaces
  - High-resolution, low cost, low jitter data transmission
  - Primary differences are in timing

- Physical bus consists of three lines: Bit Clock, Word Select, and Data

- Data rates up to a few Megabits/second
I²S (Inter-IC Sound) and Derivatives

- Used for communicating serial audio data between ICs
  - Inter-IC Sound (I²S)
  - Left Justified (LJ)
  - Right Justified (RJ)
  - Time Division Multiplexed (TDM)
- Synchronous, bi-directional serial interfaces
  - High-resolution, low cost, low jitter data transmission
  - Primary differences are in timing
- Physical bus consists of three lines: Bit Clock, Word Select, and Data
- Data rates up to a few Megabits/second
RS-232 (Recommended Standard-232)

- Point-to-point communication at slow speeds over short distances
- Two single-ended signals provide point-to-point, full-duplex communication
- Standard does not specify character encoding, data framing, or protocols
- Transmission systems:
  - Managed by Universal Asynchronous Receiver/Transmitters (UARTs)
    - Pre-determined bit rate
  - RS-232 is an inverting, single-ended high-voltage interface
  - RS-422 or RS-485 are differential interfaces
  - Or ICs can be connected directly
CAN (Controller Area Network)

- Used for system-to-system communication in Automotive, Industrial Automation, and Medical Equipment

- Serial asynchronous, multi-master, layered communication network
  - Sophisticated error detection and error handling mechanisms
  - Flexible signaling support for low-cost implementation
  - Messages are broadcast to all nodes on the network

- Physical bus is single-wire or dual-wire, and fault tolerant

- Data rates from 5 kbps to 1 Mbps
LIN (Local Interconnect Network)

- Developed in the 1990s for low-cost, low-speed automotive applications.
- Used for communications between intelligent sensors and actuators – such as window controls, door locks, rain sensors, windshield wiper controls, and climate control.
- Data rates from 1 kb/s to 20kb/s
- Single-wire transmission
- Often used as sub-network in automotive CAN networks
FlexRay

- FlexRay is a high-speed, high-reliability bus used for critical automotive applications like “x-by-wire”
- FlexRay is still being developed by the FlexRay Consortium
- FlexRay is a differential automotive serial bus
  - Data at rates up to 10 Mbps, transmitted over twisted-pair cables
How do you measure the bit rate of a serial signal?

- For all embedded-clock serial signals, you must specify bit rate
  - Capture signal
  - Measure narrowest pulse width
  - Bit rate is reciprocal of pulse width

### Minimum CAN Pulse Width vs. CAN Bit Rate

<table>
<thead>
<tr>
<th>Minimum CAN Pulse Width</th>
<th>CAN Bit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ms</td>
<td>1 Mbps</td>
</tr>
<tr>
<td>1.25 ms</td>
<td>800 kbps</td>
</tr>
<tr>
<td>2 ms</td>
<td>500 kbps</td>
</tr>
<tr>
<td>4 ms</td>
<td>250 kbps</td>
</tr>
<tr>
<td>8 ms</td>
<td>125 kbps</td>
</tr>
<tr>
<td>12 ms</td>
<td>83.3 kbps</td>
</tr>
<tr>
<td>16 ms</td>
<td>62.5 kbps</td>
</tr>
<tr>
<td>20 ms</td>
<td>50 kbps</td>
</tr>
<tr>
<td>30 ms</td>
<td>33.3 kbps</td>
</tr>
<tr>
<td>50 ms</td>
<td>20 kbps</td>
</tr>
<tr>
<td>100 ms</td>
<td>10 kbps</td>
</tr>
</tbody>
</table>

Manual measurement of CAN bit rate
Other serial bus application details to consider

- Need adequate sample rate to represent signals
  - Sample rate affects single-shot bandwidth
  - “10X” rule is a good guideline

- Signal amplitude needs to be adequate
  - Minimum digital channel input is ~ 500 mV_{pp}

- Thresholds need to be set appropriately
  - Usually centered in AC portion of signal
  - Signals like CAN have a DC offset
  - Signals like FlexRay are multi-level signals

- All packet data must be in acquisition
  - Decoder can not predict data that isn’t captured
More serial decoding details to consider

- “pre-charge” idle time needed before decoding can begin
  - At the beginning of the acquisition
  - Between packets

- Packets need to have an end:
  - Inter-packet idle time
    - 2-wire SPI clock idle time
    - LIN sync break
    - CAN INTermission field
  - Framing signal
    - RS-232 stop bits for individual characters (≥1 bit wide)
    - SPI Slave Select
    - Audio Word Select
  - Unique data value
    - RS-232 End Of Packet character in Packet mode
Troubleshooting Your Device

- **Decode of Serial Data**
  - Capture and Search for Specific Messages
  - Characterize System Timing
  - Trace Data Flow Through a Network
  - Verifying System Behavior
  - In-Depth Analysis of Network Performance
Decode of Serial Data

- **Hardware Engineers:** verify connections and adequate signal integrity for the bus to transmit data.
  - Monitor waveforms and decoded bus data values

- **Software/Firmware Engineers:** verify bus messages are being sent as expected.
  - Waveform displays are not the preferred format.

- **System Engineers:** verify system components are working together as designed.
  - Again, waveform displays are not the preferred format.

*Bus waveforms can be manually decoded… But it is tedious and error-prone.*
Automated Decode with Tektronix’ Oscilloscopes

Tektronix MSO/DPO4000 Series Automated Decode

Start

Address

[W] for Write, [R] for Read
Displayed in hex or binary

Data

Displayed in hex or binary

Stop
Event Table for Viewing Bus Traffic

- Shows decoded message content with time stamps
- View bus traffic in tabular format
- Compare with software listings
- Easy timing measurements

Tektronix MSO/DPO4000 Series Event Table
Troubleshooting Your Device

- Decode of Serial Data
- Capture and Search for Specific Messages
- Characterize System Timing
- Trace Data Flow Through a Network
- Verifying System Behavior
- In-Depth Analysis of Network Performance
Capture a Specific Message

- Even if you can easily decode messages, the message of interest probably wasn’t captured
- Need to specify messages to **capture**:
  - In the language of the serial bus standard
  - On all critical elements of the serial message
  - With full or partial specification
- Tektronix’ oscilloscopes offer serial data triggers

*Tektronix MSO/DPO4000 Series I²C Triggers*
An Example: Faulty Thermal Management System

- The product is overheating and shutting off.
- Microprocessor-controlled thermal management system should sense the product’s internal temperature and adjust the fan speed.
  - All of the circuits appear to have the correct power applied.
  - The processor is running and appears to be communicating with the sensors and the fan control module.
  - The software team is sure that the software is running as designed.
- Yet, the product is getting hot and the fan is not turning on.
Debugging with the MSO/DPO Series

- Trigger on address 18 (sensor).
- Software tries to communicate with the sensor – twice!
- No response.
- Software moves to the next address, as designed.
- Upon closer inspection of the board, a cold solder joint was found on the fan controller IC.

Faulty communication with fan controller at address 18
Troubleshooting Your Device

- Decode of Serial Data
- Capture and Search for Specific Messages
- Characterize System Timing
- Trace Data Flow Through a Network
- Verifying System Behavior
- In-Depth Analysis of Network Performance
Characterize System Timing

- Characterize timing between bus messages and system operation
  - Requires waveform displays time-correlated with decoded messages

- Characterize timing differences which occur when adding a new network node to an existing network

- Automotive application example:
  - Measure worst-case time from crash sensor output to airbag activation
  - Measure variations in timing of airbag activation with varying levels of CAN bus traffic
Characterizing System Timing with Tektronix’ Oscilloscopes

- Tektronix’ oscilloscopes provide integrated tools for characterizing timing between bus messages and system operation:
  - Time-correlated waveform displays and decoded bus messages
  - Intensity-graded infinite persistence displays to show variations in timing
Troubleshooting Your Device

- Decode of Serial Data
- Capture and Search for Specific Messages
- Characterize System Timing
- Trace Data Flow Through a Network
- Verifying System Behavior
- In-Depth Analysis of Network Performance
Trace Data Flow Through a Network

- Trace serial data flow between nodes through a network
  - Simultaneously display messages at transmitter and receiver to verify continuity and propagation delays

- Trace serial data flow between network segments separated by a gateway
  - Simultaneously display messages from multiple buses, at different speeds, or even different bus standards
Tracing Data Flow with Tektronix’ Oscilloscopes

- Tektronix’ oscilloscopes simultaneously display messages at different points in the network
  - Verify continuity and propagation delays on up to 4 buses
  - Validate network gateway operation by decoding different bus speeds & protocols
Troubleshooting Your Device

- Decode of Serial Data
- Capture and Search for Specific Messages
- Characterize System Timing
- Trace Data Flow Through a Network
  ✤ Verifying System Behavior
- In-Depth Analysis of Network Performance
Verifying System Behavior: Li-ion Battery Charger

- Battery temperature should control charge rate

- Measurement Objectives:
  - Measure response of charger to change in temperature
  - Confirm control stability during the change of current

- Measurement Challenges:
  - Decoding I²C messages to confirm correct readings
  - Capturing data over full period of current correction
Overview of Charger Operation

- Temperature
  - Sampled at the battery
  - Analog temperature reading appears at the voltage peaks
  - Negative-temperature-coefficient sensor, so higher voltage represents a lower temperature
  - As temperature decreases, we need to reduce the charging current to protect the Li-ion battery

- Power Transistor
  - Voltage and Current
  - Pulse-width modulated control of the charge current
Overview of Charger Operation

- Temperature
- Power Transistor Voltage and Current
- Decoded I^2C bus
  - Monitored by two digital lines labeled "DATA" and "CLOCK"
Charger Operation During Current Change Due to Temperature Change

- **Period**
  - Screen shows operation of charger over 4 sec to show response to a temperature change

- **Complete view**
  - Stimulus (temperature)
  - Communications (I²C)
  - Effect on the charger switching power supply (voltage and current)
  - Monitoring line in the control loop (update)

- **Response to change**
  - Change in temperature from time 1A to 1B
  - Incremental correction of the charging current
Decoded I²C Data from Initial Temperature Reading

- Magnified location of bus decode
  - Always maintains the perspective of the total trace time
- Read command (0x17)
- Original temperature value (0x0BB8)
  - Low byte first
Decoded I²C Data After Temperature Change

- Magnified location at next sample
- Occurs after the temperature change is measured by the battery protection electronics
- Read command (0x17)
- Updated temperature value (0x08D4)
Verify Gradual Current Reduction

- Magnified view
  - Shows gradual current adjustment

- Current updates
  - Each "Update" pulse incrementally corrects the current
  - Smooth adjustment from high to low current level
    - ~10 msec intervals
    - ~200 msec transition
    - ~2 A to ~0.25 A change
Conclusion: Li-ion Battery Charger Example

- Achieved objectives:
  - Measured response of charger to change in temperature
    - Current changed about 200 milliseconds after temperature change
    - Decoded I²C data correctly showed the temperature
  - Confirmed control stability during the change of current
    - Current changed smoothly from 2 A to 0.25 A

- Solution to measurement challenges:
  - I²C decode capability of the MSO/DPO Series is valuable to observe systems
  - With deep record length, a single acquisition showed full period of change while also showing the detailed operation of the switching power supply
Troubleshooting Your Device

- Decode of Serial Data
- Capture and Search for Specific Messages
- Characterize System Timing
- Trace Data Flow Through a Network
- Verifying System Behavior
- In-Depth Analysis of Network Performance
In-Depth Analysis of Network Performance

- Locate and analyze signal integrity problems with eye diagrams
- Characterize different oscillator tolerances and propagation delays between nodes for synchronizing the network
- Monitor bus utilization to ensure efficient use of the network
Eye Diagram Analysis with Tektronix’ Oscilloscopes

- Quickly locate noise caused by jitter, amplitude aberrations, spikes and glitches
  - Eye diagram shows changes in amplitude and jitter in the CAN bus signal
  - Measure amplitude and jitter with cursors

Tektronix DPO7000 Series
Characterizing Oscillator Tolerance and Propagation Delay with a Tektronix’ Oscilloscope

- **Oscillator tolerance of a CAN node**
  - Specify the specific ID for trigger condition
  - Result will include ACK and without ACK bit
  - With ACK bit, shows the impact of receiving CAN node oscillator tolerance on transmitting node

- **Propagation Delay**
  - Connect two channels to any two CAN nodes
  - Result is directly available

**Tektronix DPO7000 Series with TDSVNM option**
Monitoring CAN Traffic for Bus Utilization with a Tektronix’ Oscilloscope

- Measure at specific ID, error frame or overload frame
- Specifies percentage of time traffic present in the CAN bus
- Type of traffic can be analyzed
  - Frame count

Tektronix DPO7000 Series with TDSVNM option
Summary

- Serial buses are pervasive, creating a unique set of measurement and analysis needs
- Making measurements needs to be easier, faster, and more accurate
- Requires an oscilloscope with triggering, decoding, and analysis tools for serial protocols

*Tektronix’ oscilloscopes offer automated decode, trigger, search and analysis for pervasive serial buses*
Debugging Serial Buses with the MSO/DPO Series

**Automated Decode, Trigger and Search**

**MSO/DPO2000 Series**
- 100 MHz and 200 MHz
- 1 M record length
- Wave Inspector
- Supported serial buses:
  - I²C
  - SPI
  - CAN
  - LIN
  - RS-232/422/485

**MSO/DPO3000 Series**
- 100 MHz to 500 MHz
- 5 M record length
- Wave Inspector
- Supported serial buses:
  - I²C
  - SPI
  - CAN
  - LIN
  - RS-232/422/485
  - I²S/LJ/RJ/TDM

**MSO/DPO4000 Series**
- 350 MHz to 1 GHz
- 10 M record length
- Wave Inspector
- Supported serial buses:
  - I²C
  - SPI
  - CAN
  - LIN
  - FlexRay
  - RS-232/422/485
  - I²S/LJ/RJ/TDM
In-Depth Analysis of Serial Buses with the DPO7000 Series

Automated Decode, Trigger, Search and Eye Diagram Analysis

DPO7000 Series
- 500 MHz to 3.5 GHz
- Up to 400 M record length
- CAN propagation delay and oscillator tolerance
- CAN bus traffic monitoring
- Supported serial buses:
  - I²C
  - SPI
  - CAN
  - LIN
  - FlexRay
  - UART/RS-232
Tektronix Signal Generators
Evaluation of CAN Bus ECU Performance

- **Signal Generator: AFG3252**
  - Replicates Sensor’s Signal

- **Oscilloscope: MSO4104**
  - Validates ECU’s CAN bus message
  - Measures latency of sensor/ECU chain
Tektronix Signal Generators
Creating Low-Speed Serial Signals

ArbExpress to Import, Edit, Save, and Export

MSO4104 to Capture

AFG3252 to Generate

Low-Speed Serial Transmitter

Slave Device Under Test
Serial Bus Analysis Information

- Application notes
- Fact sheets
- Recommended test equipment
- Webinars

Visit [www.tektronix.com/serialdebug](http://www.tektronix.com/serialdebug)
Thank You for Attending