ECE3140 / CS3420
Embedded Systems

Lecture 6. Embedded Bus Protocols

Prof. José F. Martínez

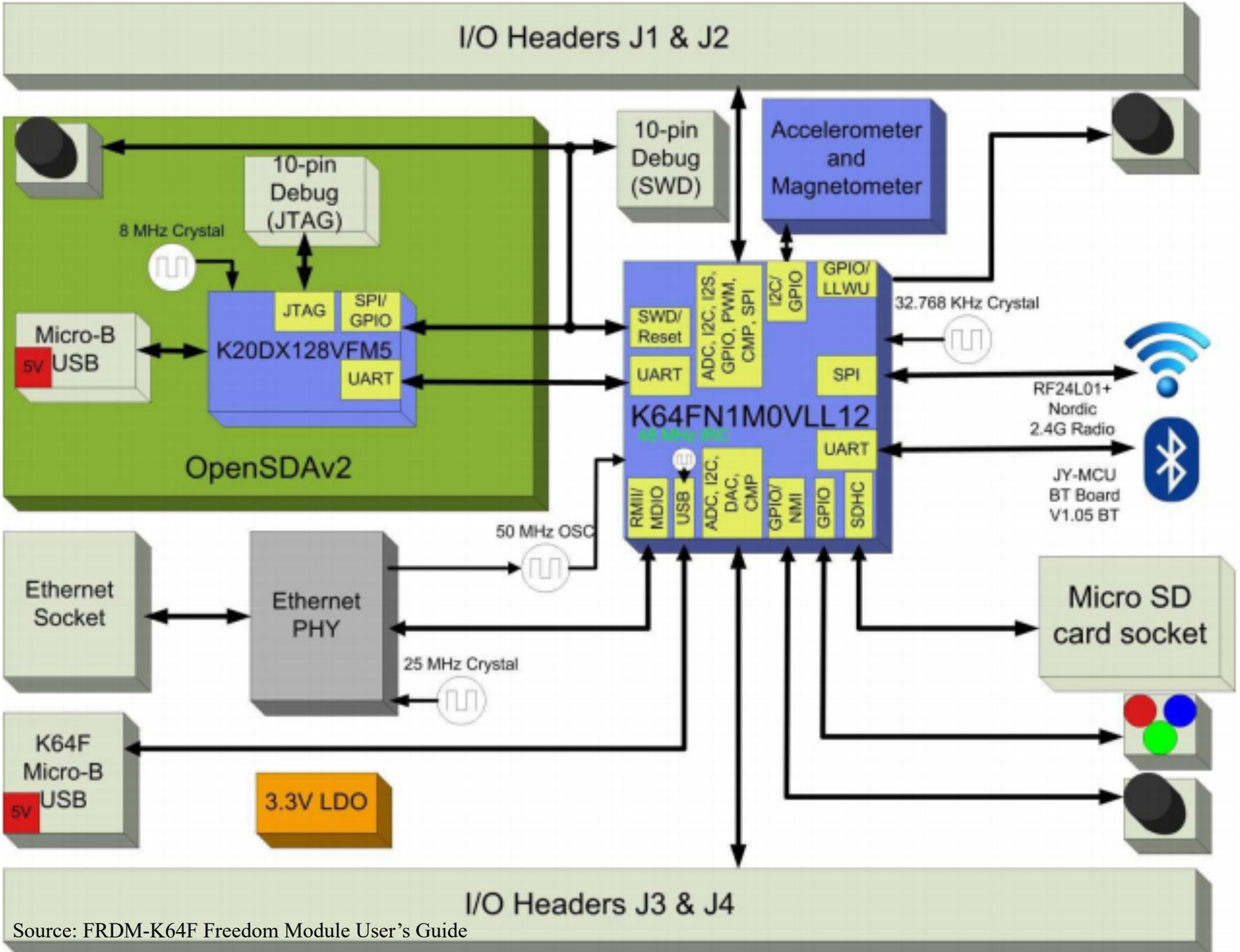
Cornell University

Outline

- IO Interfaces beyond GPIO
  - Often, with dedicated communication controllers
- Design space for bus communication protocols
  - Serial vs. parallel
- CAN bus protocol
- I^2C (Inter-Integrated Circuit) bus protocol

References

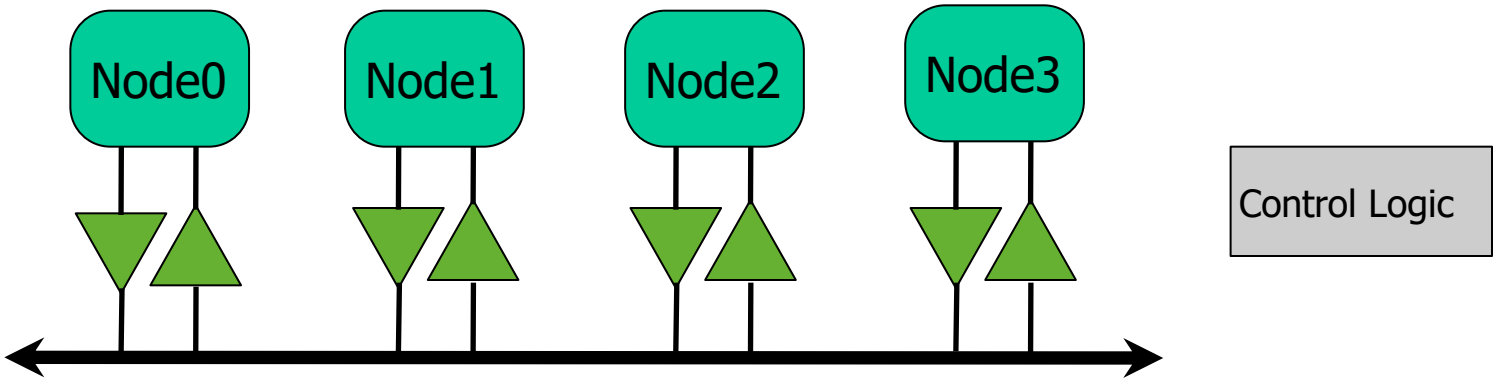
- General discussion on serial communication
  - Chapter 8 of Embedded Systems Fundamentals with ARM Cotext-M based Microcontrollers
- CAN protocol
  - Introduction to the Controller Area Network (CAN)
    - http://www.ti.com/lit/an/sloa101b/sloa101b.pdf
  - Wikipedia: https://en.wikipedia.org/wiki/CAN_bus
- I2C protocol
  - https://www.i2c-bus.org/
  - NXP bus specification
    - https://www.nxp.com/docs/en/user-guide/UM10204.pdf



Source: FRDM-K64F Freedom Module User's Guide

Bus

- "A communication system that transfers data between components inside a computer, or between computers" (Wikipedia definition)
  - Usually among multiple nodes
- Components
  - Set of wires connecting all the nodes
  - Special circuits for sending and receiving bus data
  - Control logic that decides who is allowed to send
    - May be integrated into nodes



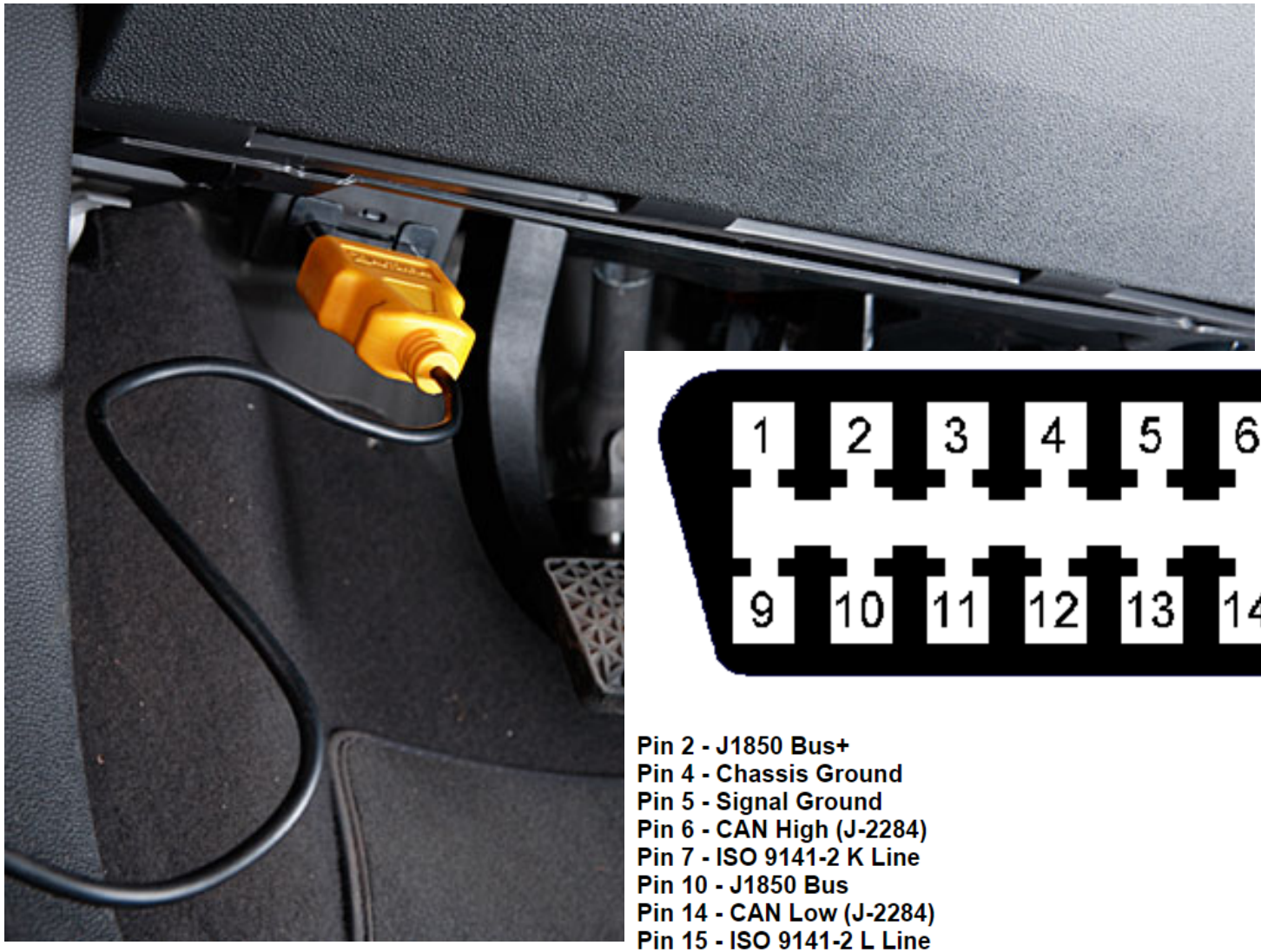
Design Questions

- How to transmit bits?
  - Serial vs. parallel
  - Clock vs. no clock
  - Bit encoding
    - NRZ (Non-Return-to-Zero), RZ (Return-to-Zero), etc.
- How to know who (sender/receiver) / what (message type)?
  - Address, identifier, etc.
- How to handle contentions?
  - Arbitration
- How to handle errors?
  - Acknowledgements, error detection/correction code, etc.

CAN Bus

- CAN = “Controller Area Network”
- A vehicle bus standard developed in 1983. First used in BMW 8 series in 1988.
- Used in many embedded environments
  - Automotive: Engine, transmission, airbags, ABS, cruise control, power steering, audio system, power locks and windows, etc.
  - Cycling: Electronic gear shift
  - Industrial: Plant automation

On-Board Diagnostics (OBD-II)

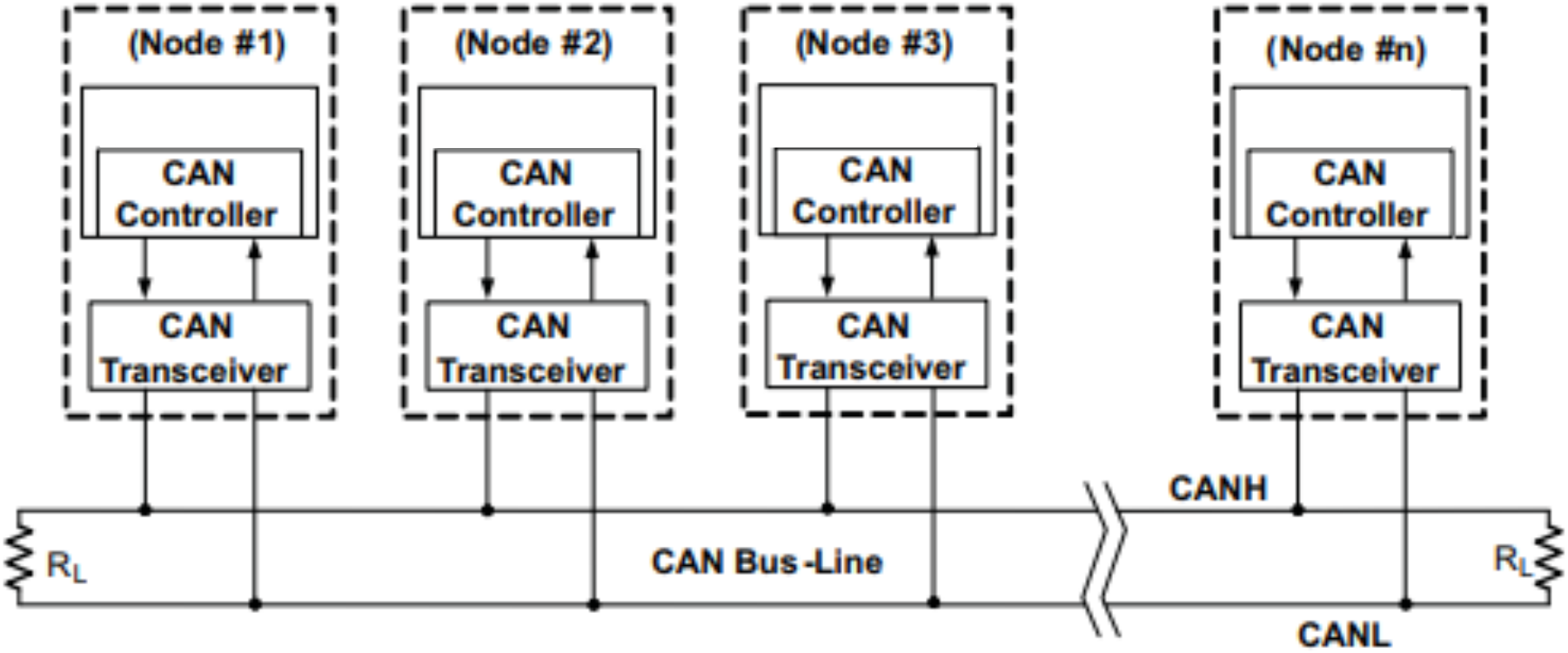


Pin 2 - J1850 Bus+
Pin 4 - Chassis Ground
Pin 5 - Signal Ground
Pin 6 - CAN High (J-2284)
Pin 7 - ISO 9141-2 K Line
Pin 10 - J1850 Bus
Pin 14 - CAN Low (J-2284)
Pin 15 - ISO 9141-2 L Line
Pin 16 - Battery Power

CAN: Basic Characteristics

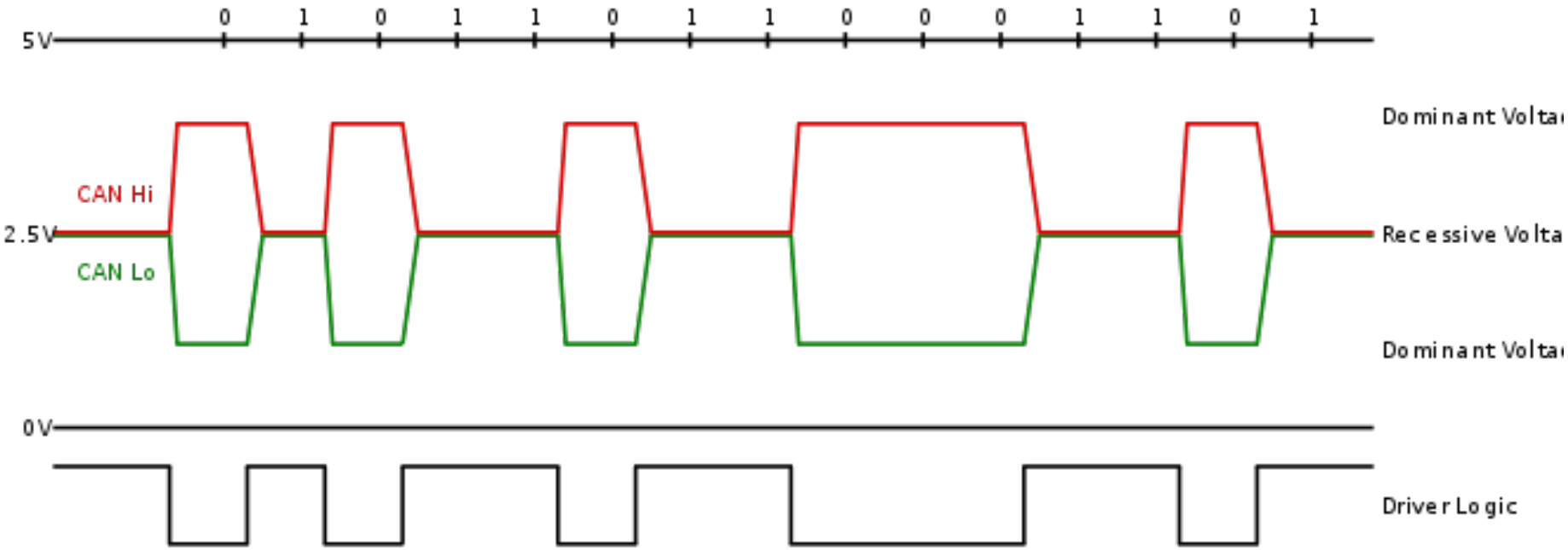
- Serial data bus – one bit at a time
- No clock line – data line is “self-clocked”
- Multi-master
  - Anyone can initiate transmission on the bus
  - Broadcast bus
- No “device address”, rather “Message ID”
- Differential signaling – no common ground voltage
  - Twisted pair with terminators for resilience and simplicity
- Wired-AND for arbitration
- ACK
- 1 Mbits per second

CAN Bus Organization



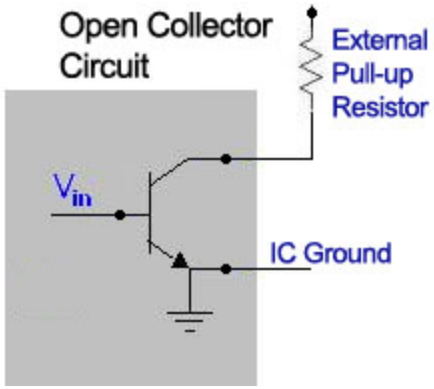
Source: ti.com

Bit Encoding



Source: Wikipedia

- Wired-AND is used as a way for bit-wise arbitration
  - Logical 0 is “dominant”
  - Logical 1 is “recessive”
  - If there is anyone transmitting ‘0’, the bus will show ‘0’

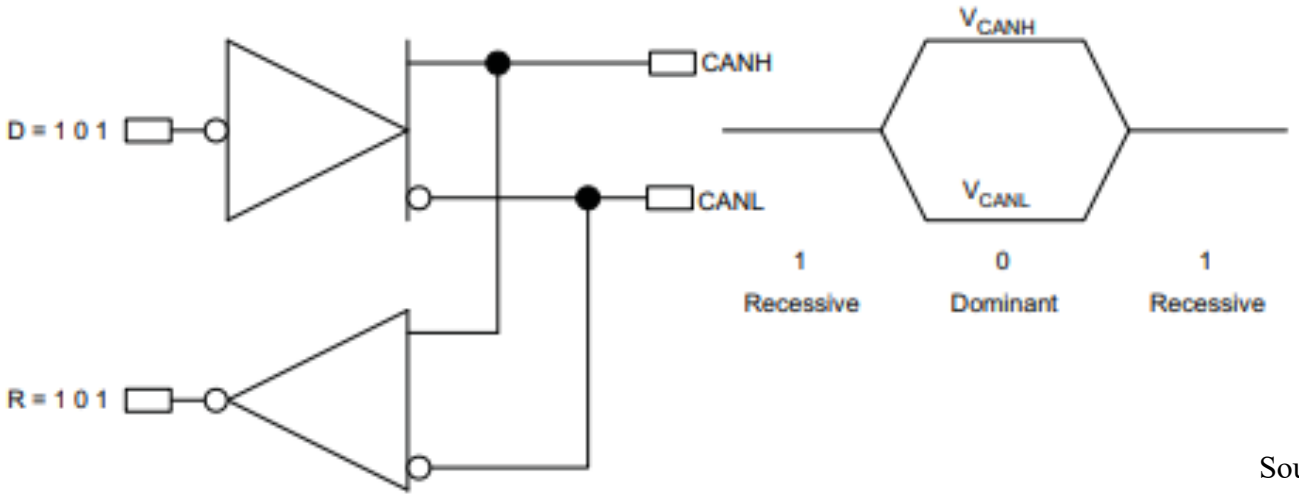


Source: National Instruments

Pull up CAN Lo to 2.5V

Transmitter / Receiver

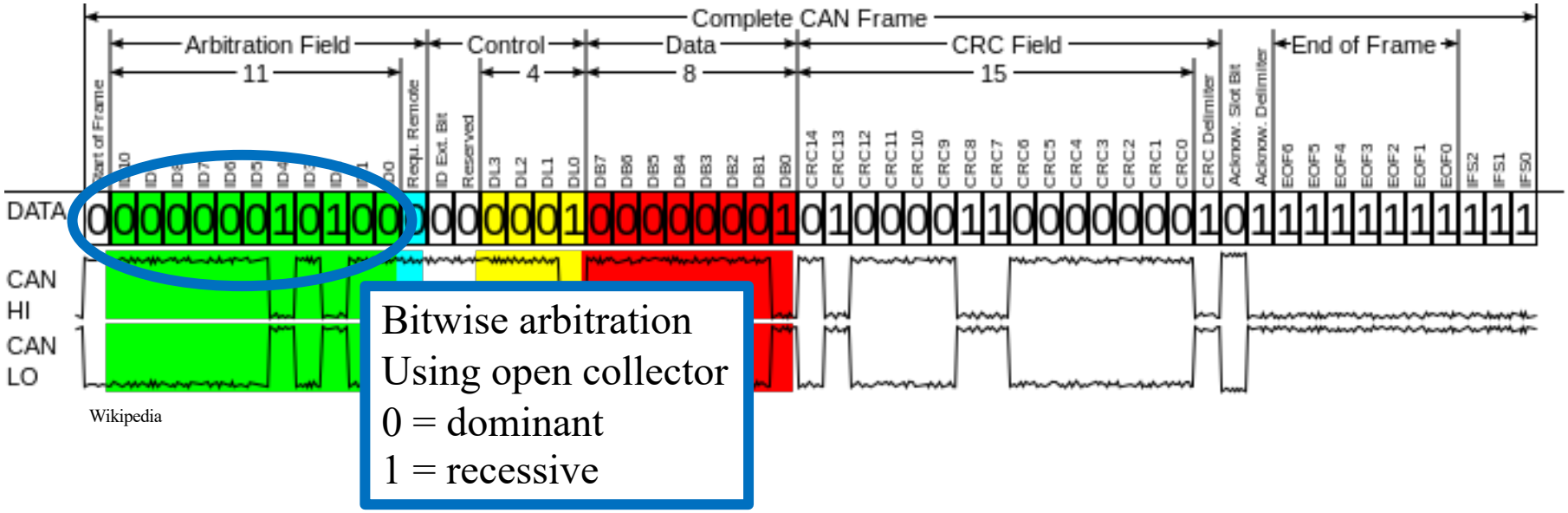
- A node can simultaneously send and receive a bit
- A node reads the bus in each cycle
  - Check to see if the received value matches what it is transmitting



Source: ti.com

Frame: SoF and Base Message ID

- Start-of-Frame (SoF)
  - Put ‘0’ on the bus after an Inter-Frame-Space (‘1’s)
- 11-bit base message ID

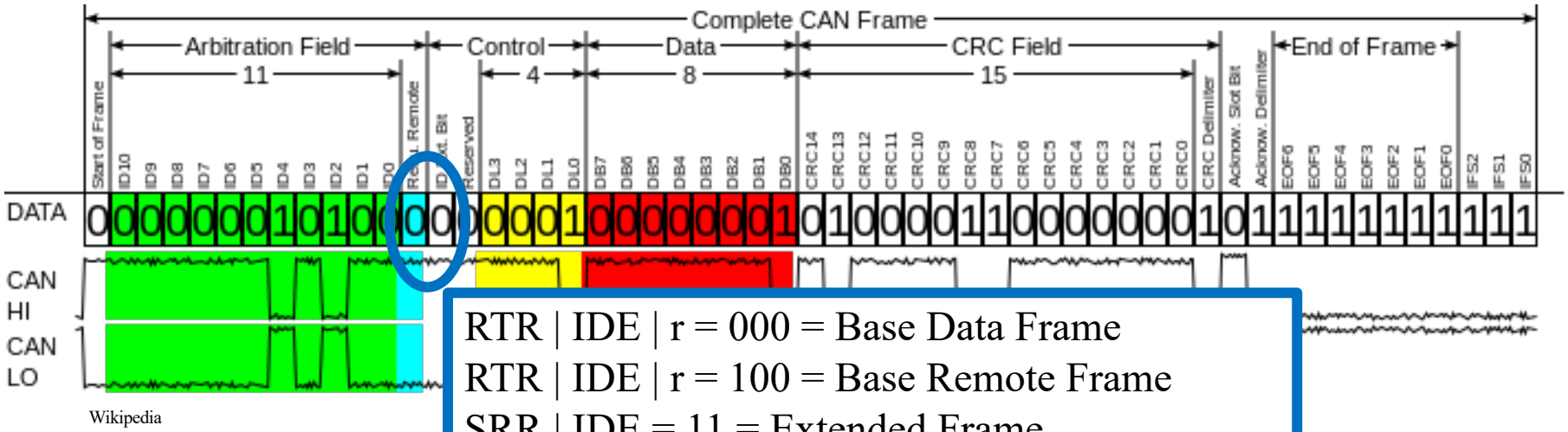


Arbitration

- Each node monitors the value on the bus while sending a bit
  - If the value does NOT match the one that it sends, the node backs off and try again later
  - Implication: the node sending ‘0’ (dominant) wins
- Example
  - Device 1: Sends ‘1’ ‘0’ ‘1’
  - Device 2: Sends ‘0’ ‘1’ ‘1’
  - Device 3: Sends ‘0’ ‘0’ ‘1’

Frame: RTR and IDE

- RTR: Remote transmission request
  - ‘0’: Data frame – contains data for transmission
  - ‘1’: Remote frame – requests a transmission of a specific ID
- IDE: Identifier extension bit



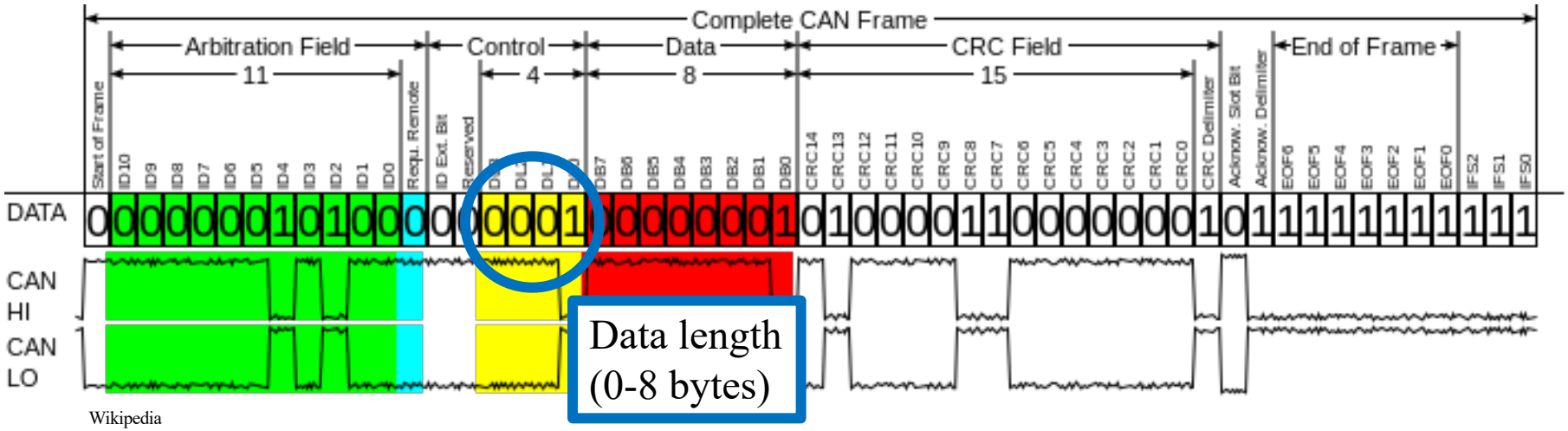
Wikipedia

RTR | IDE | r = 000 = Base Data Frame
RTR | IDE | r = 100 = Base Remote Frame
SRR | IDE = 11 = Extended Frame

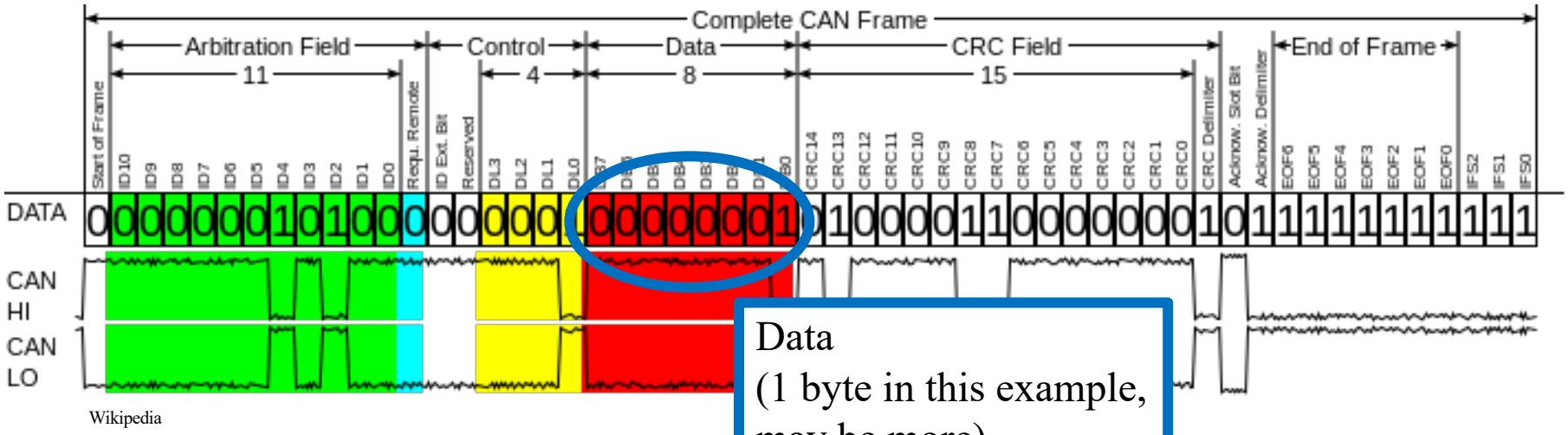
Extended frame: 18 additional ID bits | RTR | 00

Data Lengths

- DLC: Data length code
  - Number of bytes of data (0 to 8 bytes)

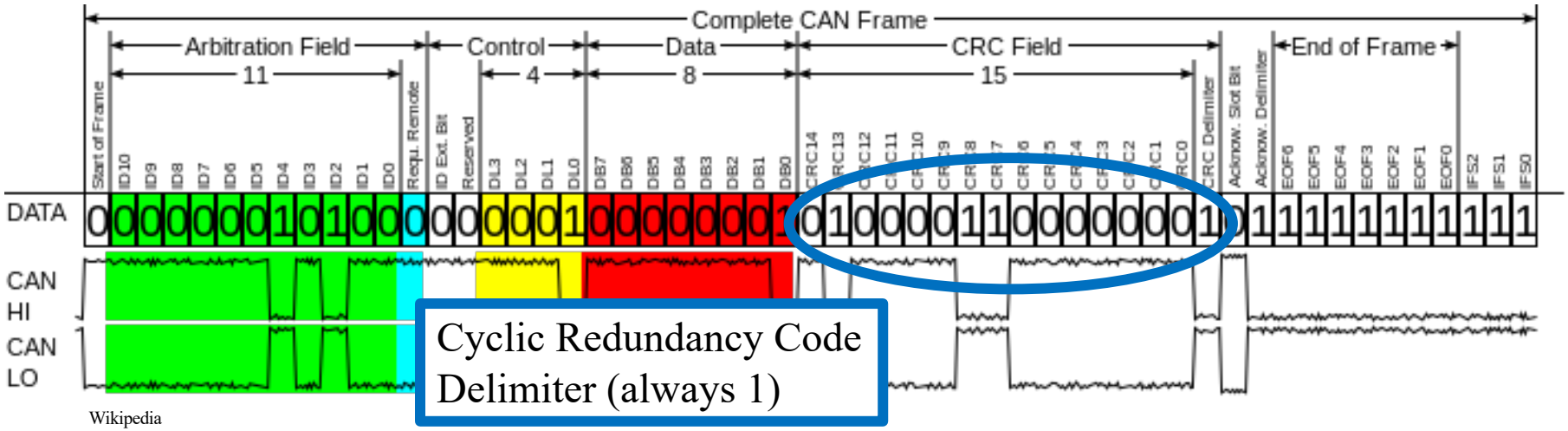


Frame: Data


Complete CAN Frame
Arbitration Field
11
Control
4
Data
8
CRC Field
15
End of Frame
Start of Frame
ID10
ID9
ID8
ID7
ID6
ID5
ID4
ID3
ID2
ID1
ID0
Requ. Remote
ID Ext. Bit
Reserved
DL3
DL2
DL1
DL0
CRC14
CRC13
CRC12
CRC11
CRC10
CRC9
CRC8
CRC7
CRC6
CRC5
CRC4
CRC3
CRC2
CRC1
CRC0
CRC Delimiter
Acknow. Slot Bit
Acknow. Delimiter
EOF6
EOF5
EOF4
EOF3
EOF2
EOF1
EOF0
IFS2
IFS1
IFS0
DATA
CAN HI
CAN LO
Wikipedia
Data
(1 byte in this example, may be more)

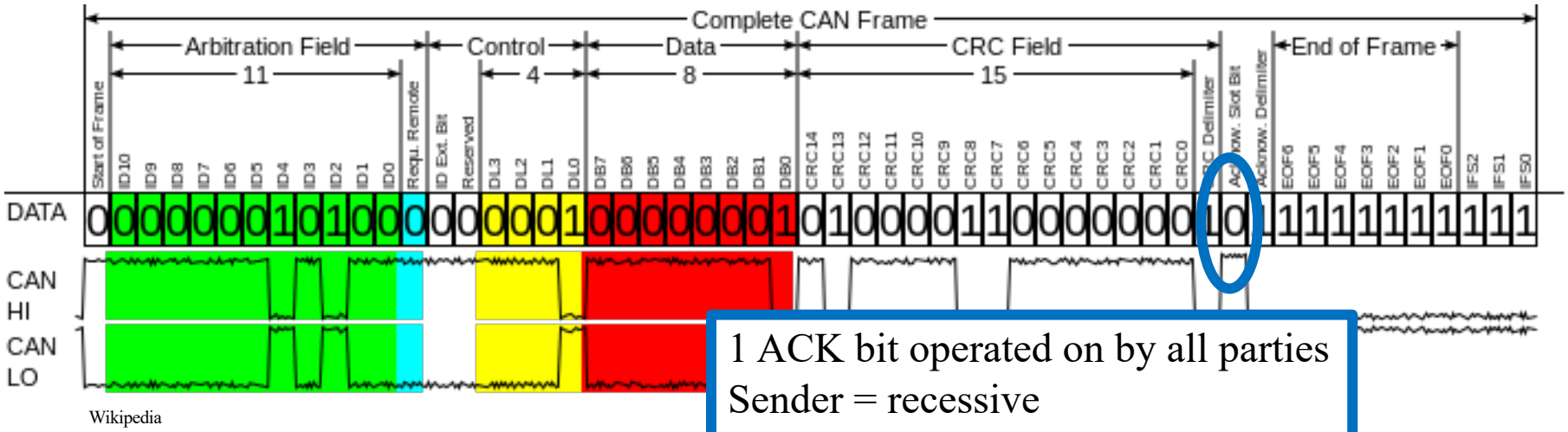
Frame: CRC Check

- CRC: Cyclic Redundancy Code
  - If an error is detected, a node sends an error frame (six consecutive ‘0’ on the bus) and other nodes follow with another error frame
  - The original transmitter sends the message again

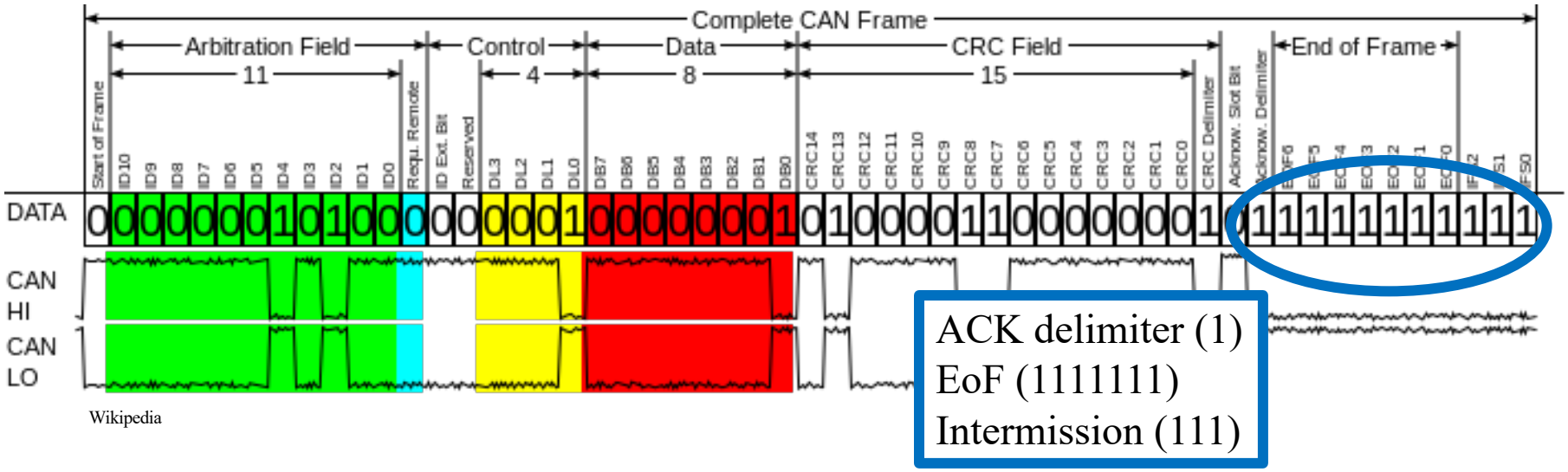


Frame: ACK

- ACK: Acknowledgement
  - All nodes that received data without a CRC error sends ‘0’
  - The transmitter detects an error if there is no ACK



Frame: Epilogue



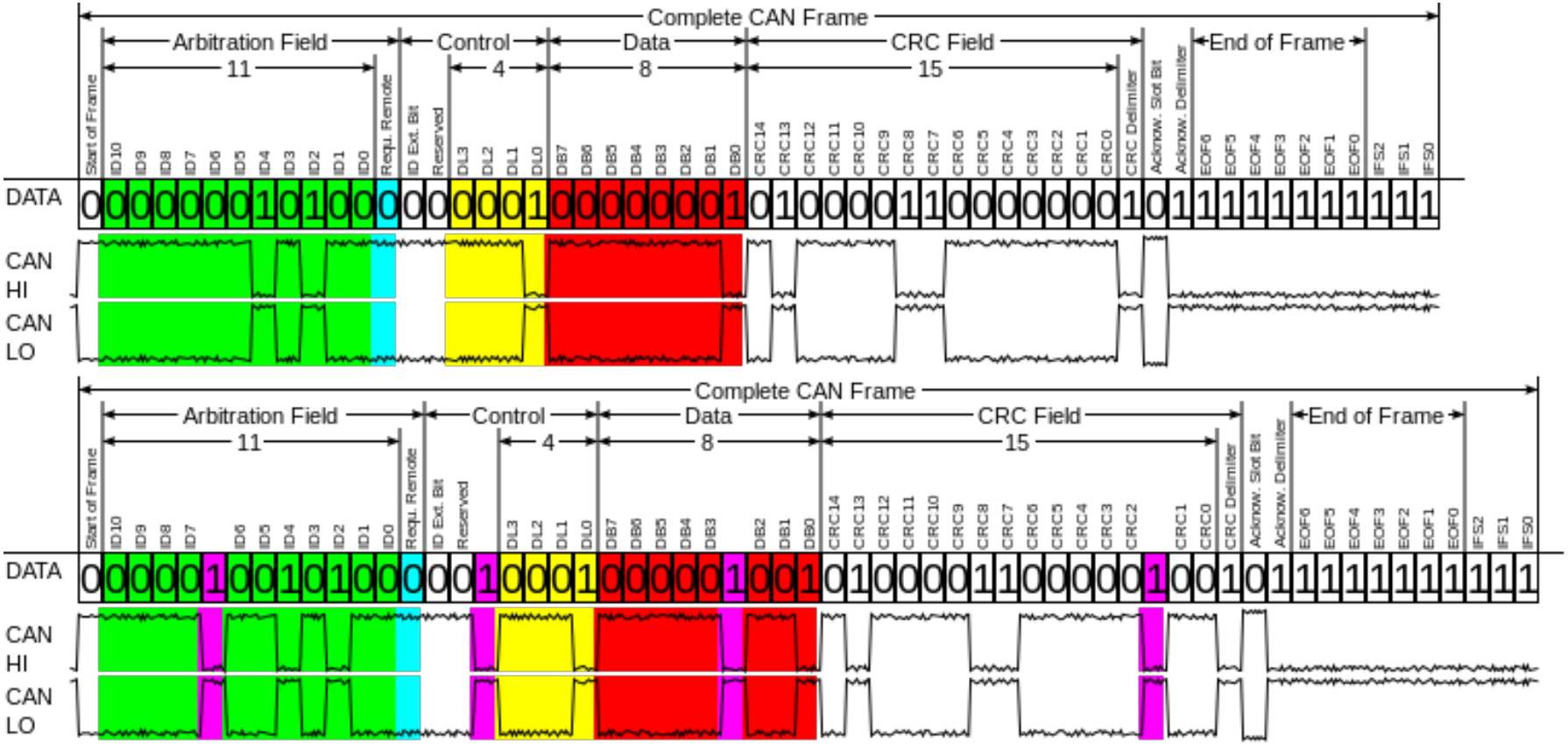
Wikipedia

Synchronization

- How do nodes know when to read the bus without an explicit clock signal?
  - Need to synchronize their internal clocks
- Nodes synchronize on the 1-to-0 transition at the start of frame (SOF) bit
- During transmission of a frame, the nodes re-synchronize when the value on the bus changes
- What if a node needs to transmit many consecutive 1s or 0s?

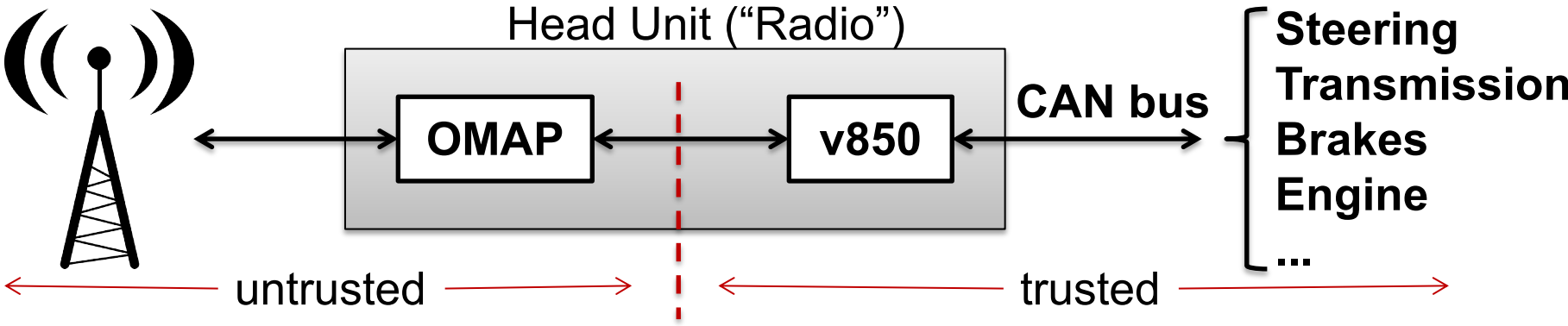
Bit Stuffing

- After five bits with the same value, insert the opposite value
  - The stuff bits are automatically removed at the receiver



Wikipedia

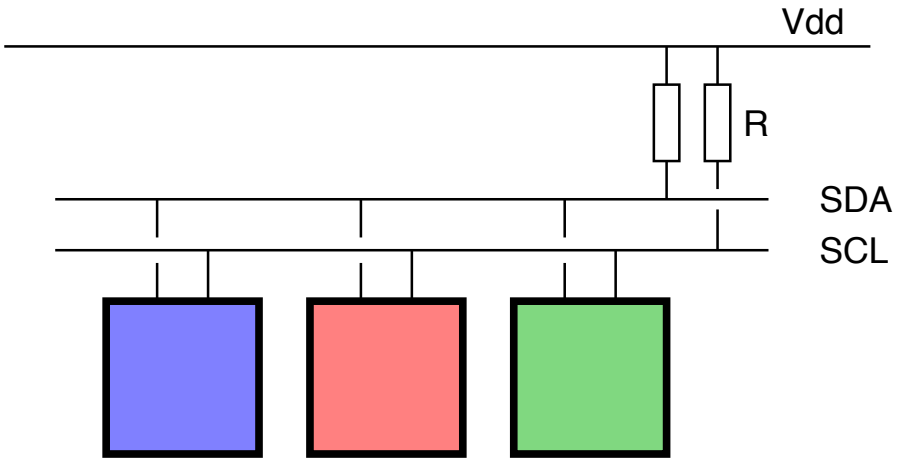
Attack on Jeep [Miller & Valasek 2015]



- Head unit runs mainly on OMAP chip
- OMAP communicates w/ v850 chip
  - For remote engine start, remote door unlock
- All software updates **unsigned** & performed via head unit
  - Including updating v850 software
- After compromising OMAP via cell connection, attackers able to update v850 & control vehicle via CAN bus

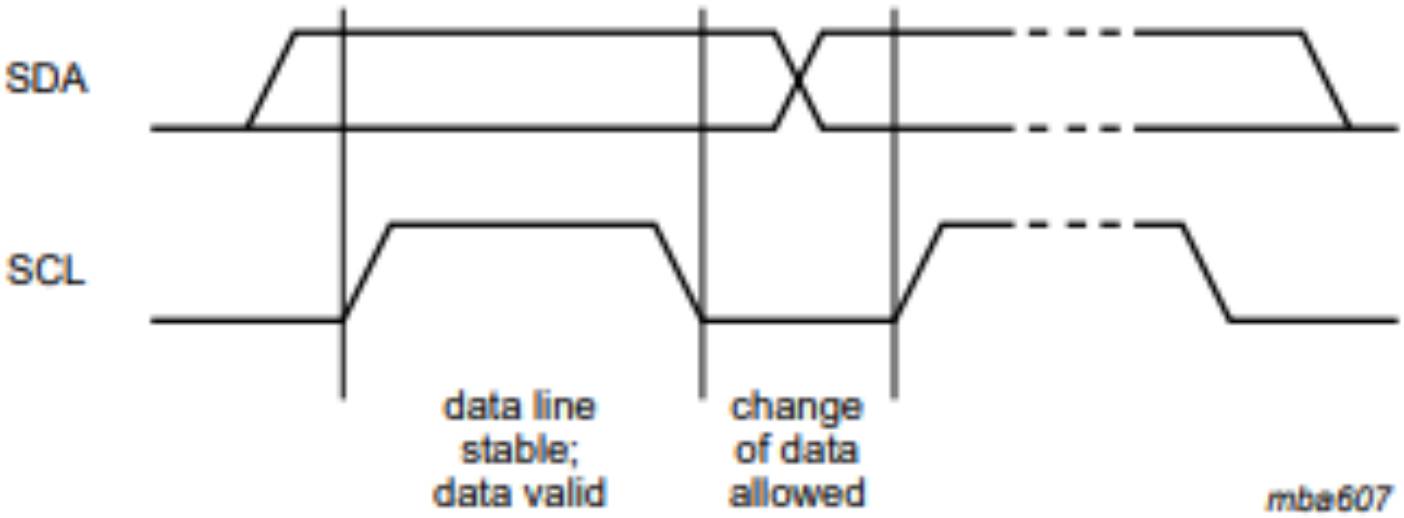
I²C Bus

- I²C = “Inter-integrated circuit”
  - Designed by Philips in the early ’80s to allow easy communication between components on the same circuit board
- Used in “cheap” embedded environments
- Multi-master, multi-slave bus
- Serial data and clock lines
  - SCL driven by master (normally)
  - SDA driven by master or slave
- Wired AND connections
  - ‘0’ wins over ‘1’
- No CRC check
- 100 kbits per second (original)
  - 400 kbit / 3.4Mbit mode available



Data Validity

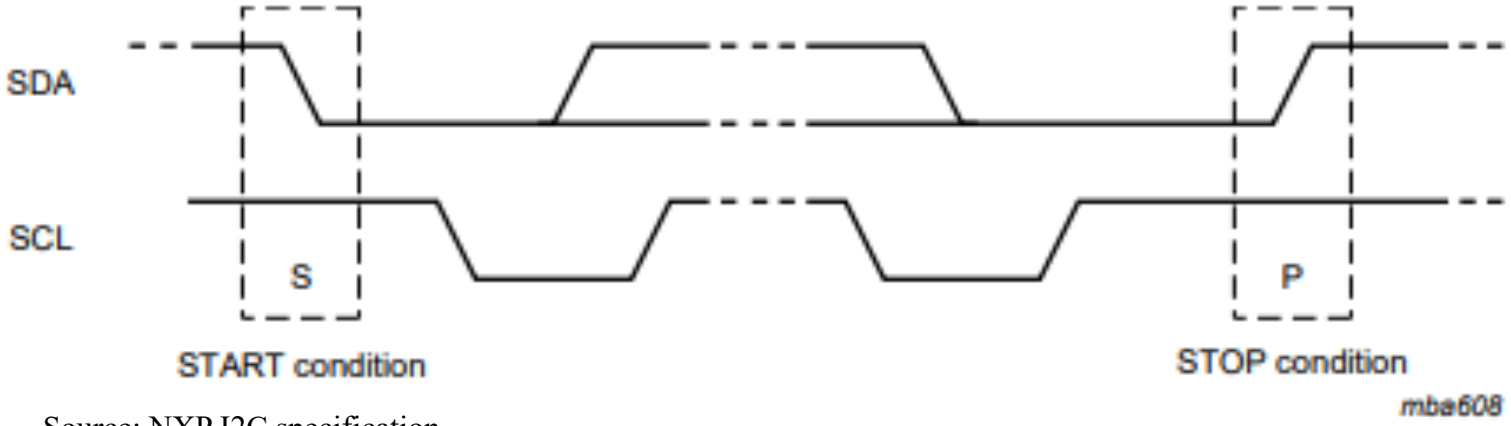
- Data set when SCL low
- Latched when SCL high



Source: NXP I2C specification

START and STOP

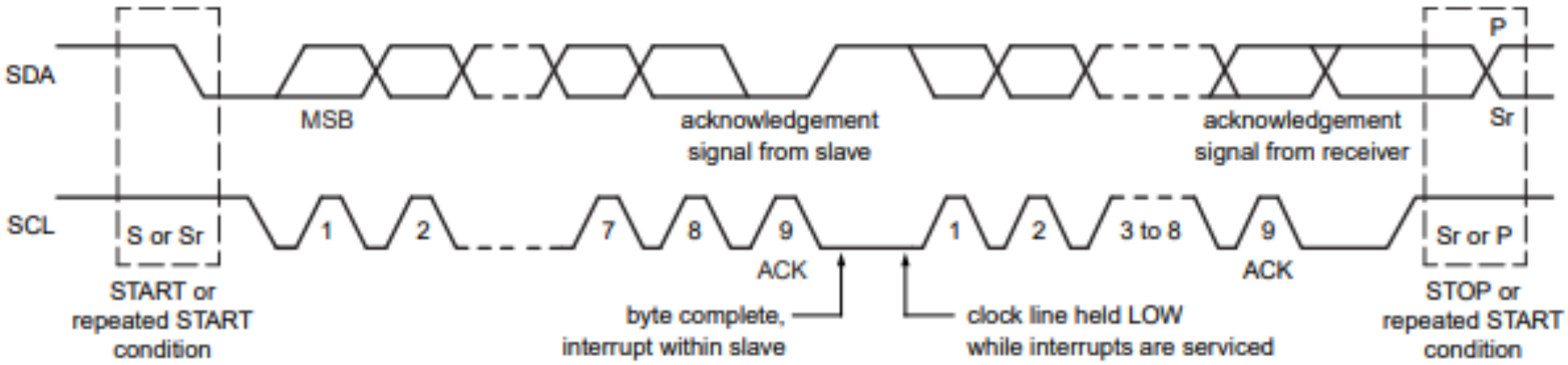
- Master signals START
  - Change SDA from high to low while SCL high
- Bus considered busy until STOP
  - STOP indicated by changing SDA from low to high while SCL high
- Same master can send multiple messages to different slaves
  - Interleaving START signals (w/o STOP)



Source: NXP I2C specification

Arbitration and Data Transmission

- 7-bit address identifies a slave
  - Wired-AND for arbitration; first to send ‘1’ while another one is sending ‘0’ loses
- R/W bit follows
  - Read from slave = 1; Write to slave = 0
- Every 9th bit is ACK (drive low)
  - 1st time slave, then data receiver
- Slaves can “clock-stretch” (drive SCL low) to force pause



Source: NXP I2C specification