These slides are partially taken from the on-line course shown below. Please refer to the reported link for more information. Please integrate the information about different communication mechanisms and protocols with further research, clear and complete documents can be easily found in literature.

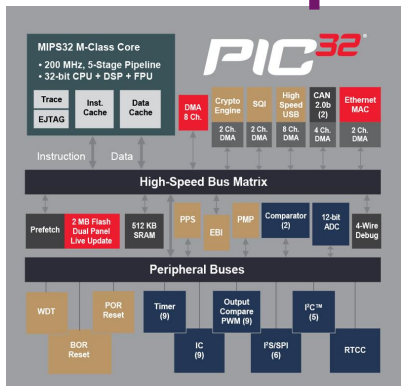
These materials produced in association with Imagination.
Join our University community for more resources.
community.imgtec.com/university

Section 1:
Communication Concepts

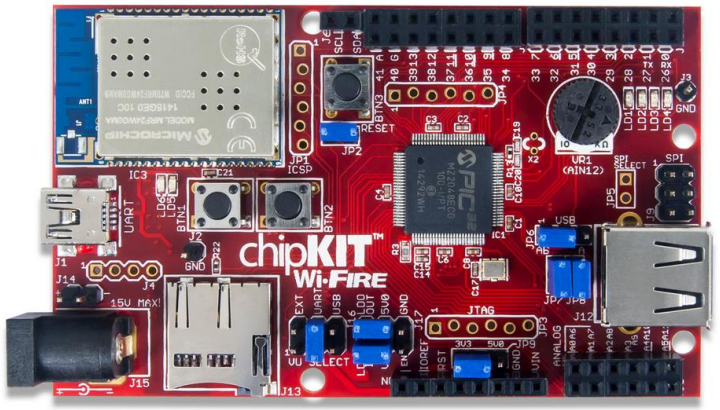
Overview of Embedded System Communications

How far does the message go?

On-Chip



On-board



In-System

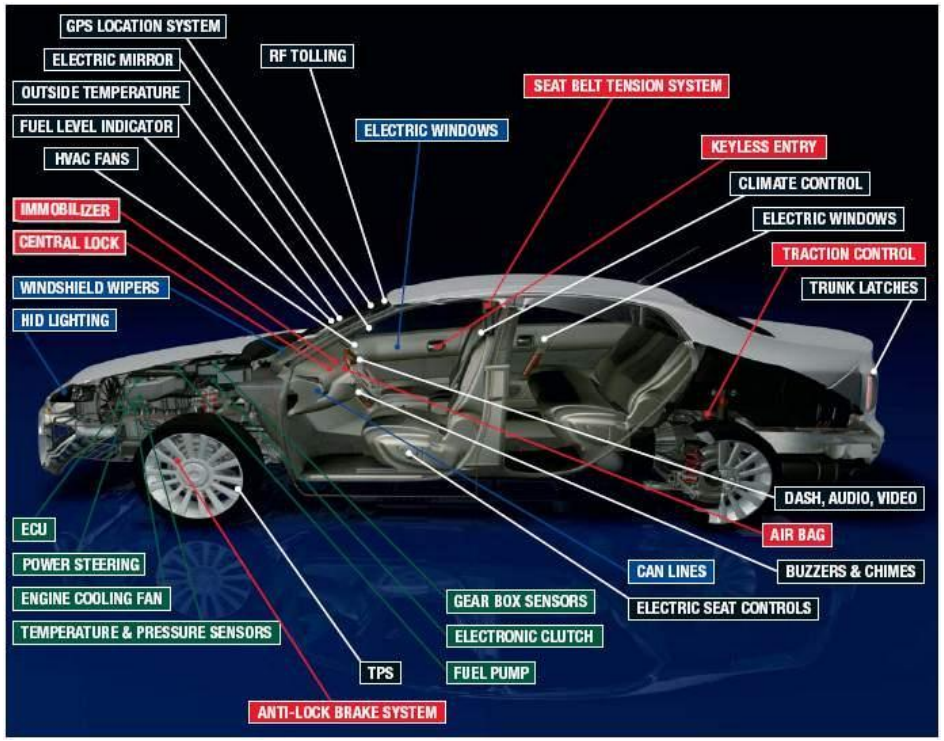


Image courtesy of AVX, Inc.

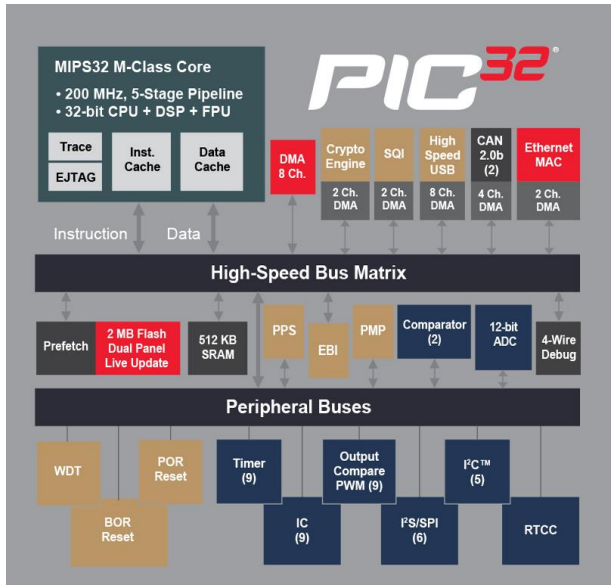
External

Image courtesy of Emerson Electric, Inc.

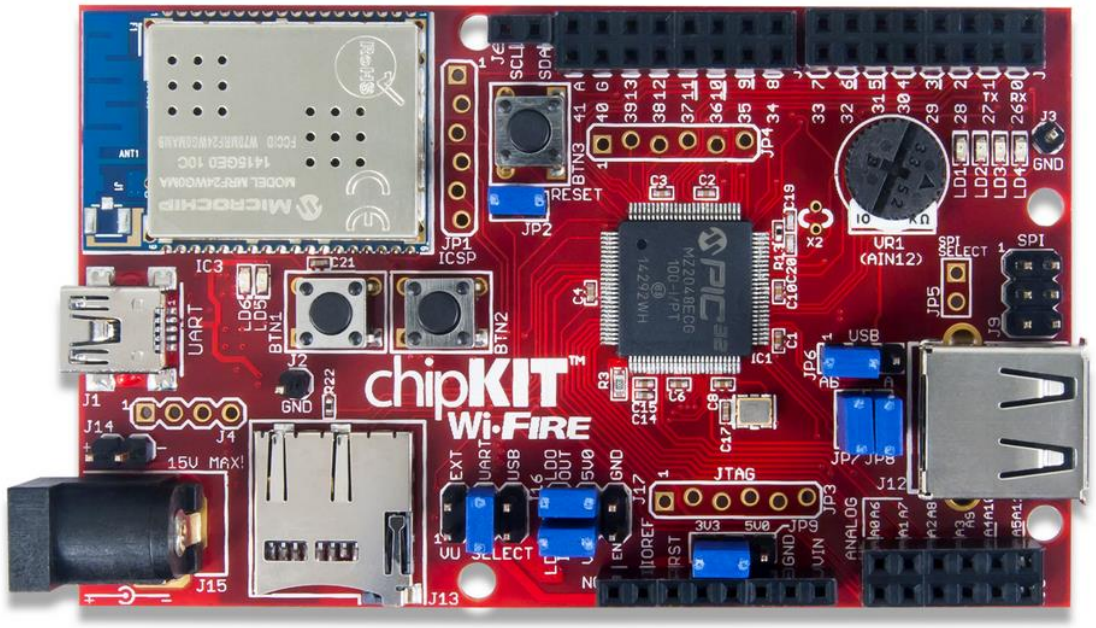
Communication Scenarios

Different constraints for different system sizes

- On-chip
  - Speed most important, minor area constraints
  - Typical solution: use parallel buses to send data 8, 16 or 32 bits at a time.



- On-board, board-to-board
  - More signals -> more pins on IC package ($$) -> larger, heavier board ($$)
  - If too slow, use parallel bus or wider serial bus, or raise clock speed
  - Typical solutions: serial buses to send data one bit at a time. SPI, I^{2}S, I^{2}C, UART



Communication Scenarios

Different constraints for different system sizes

- External – box-to-box, or system-to-system
  - More signals -> larger cable ($$) -> heavier, larger system
  - Going outside the box makes communications more vulnerable to noise
    - Add error control: detection, acknowledgment, correction
  - Typical wired solutions: serial buses to send data one bit at a time.
    - USB (Universal Serial Bus), I^2C, UART, CAN, FlexRay
  - If fast communications are important, use more bits or raise clock speed
    - Ethernet, USB 1.2/2.0, USB 3.0
  - If portability is important, use wireless transmission
    - WiFi (802.11), LTE (cellphone network), 802.15.4

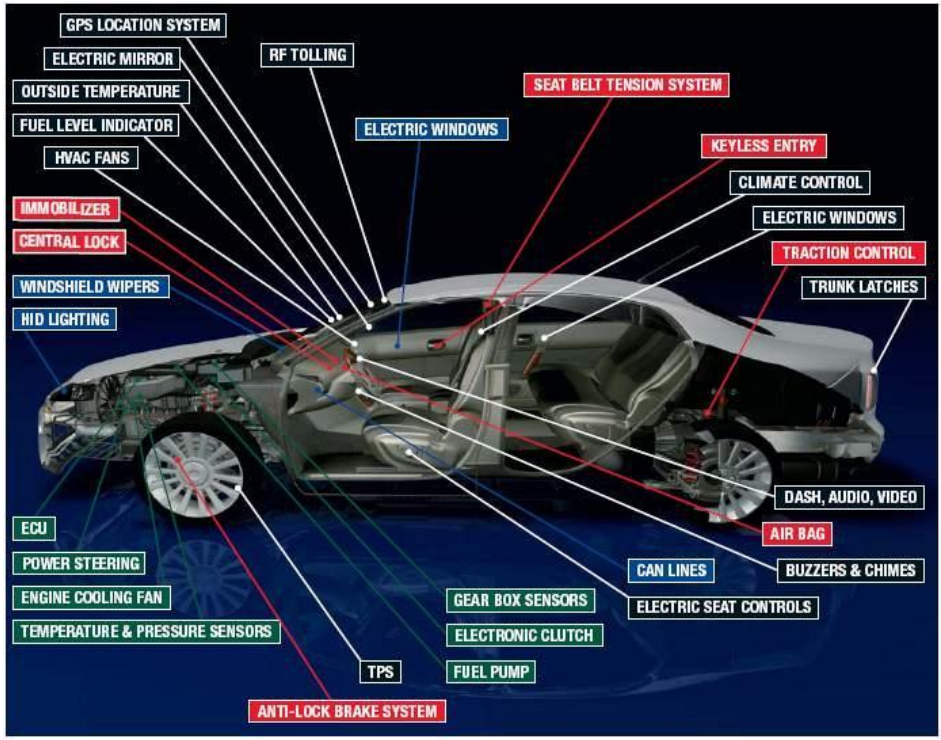
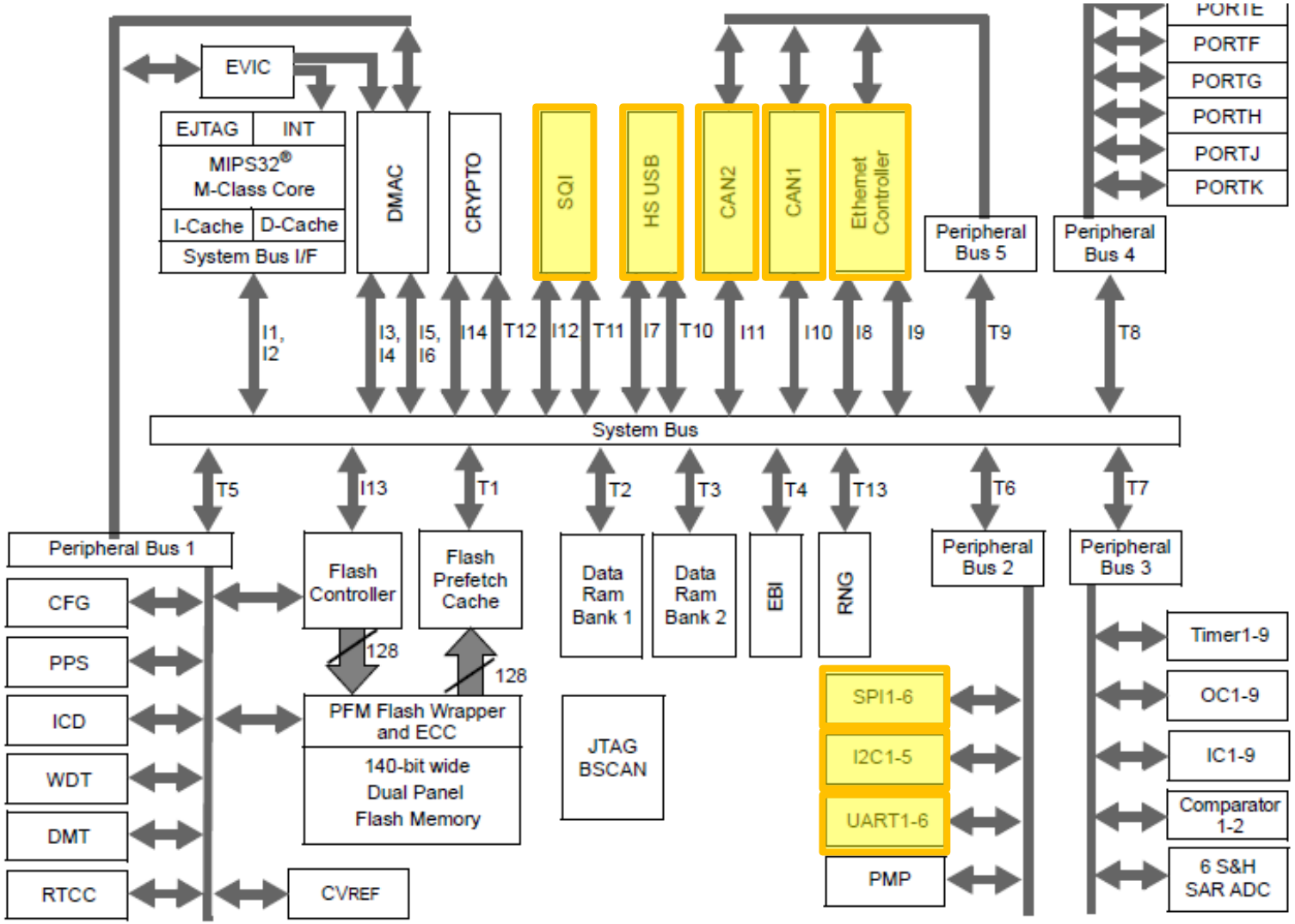


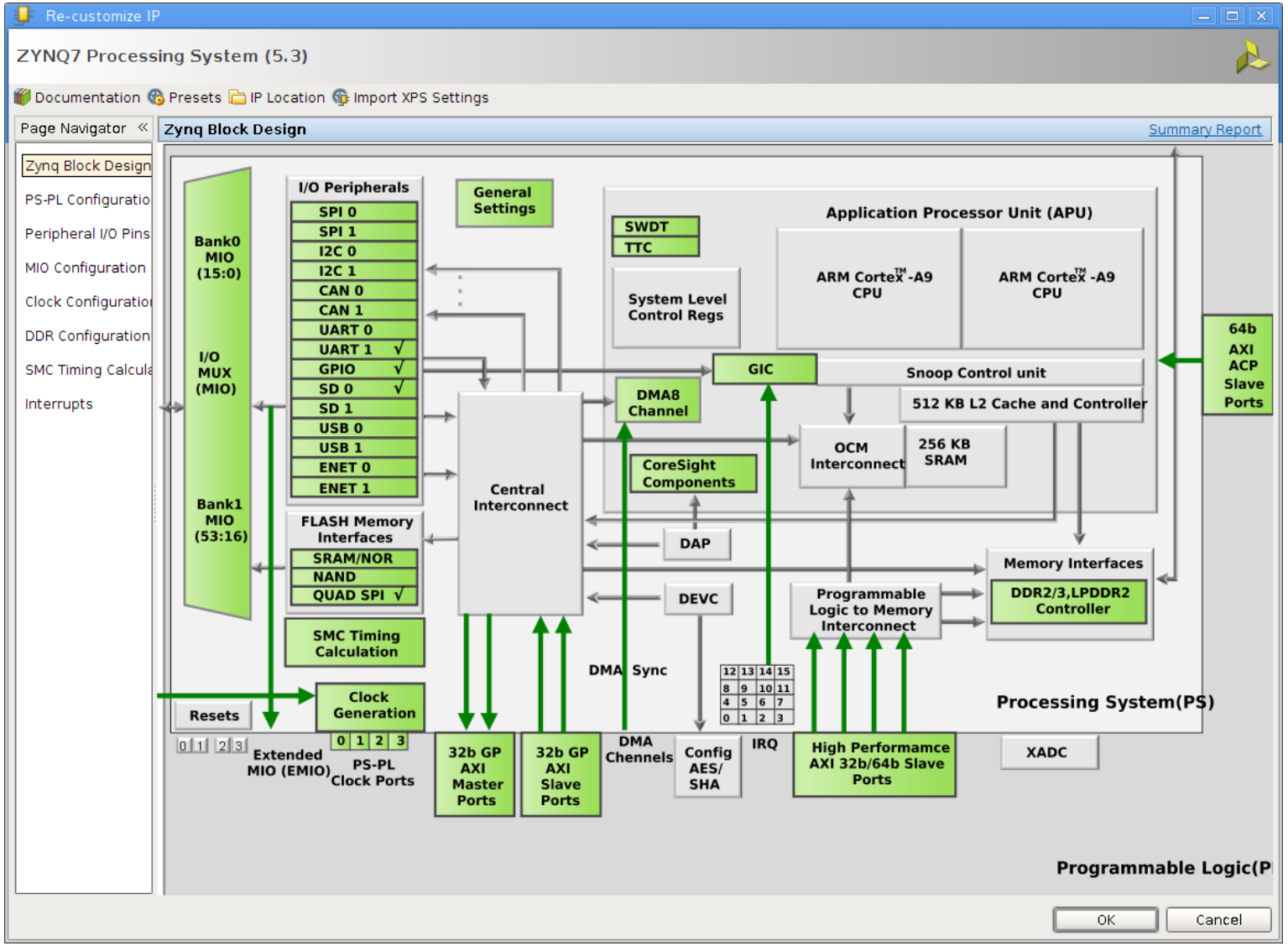
Image courtesy of AVX, Inc.

Image courtesy of Emerson Electric, Inc.

PIC32MZ EF Peripherals for Communication

- SPI
- SQI
- UART
- Hi-Speed USB with On-The-Go
- I²C
- CAN
- Ethernet





Protocol Stack Concepts

All nodes must follow the same or compatible rules

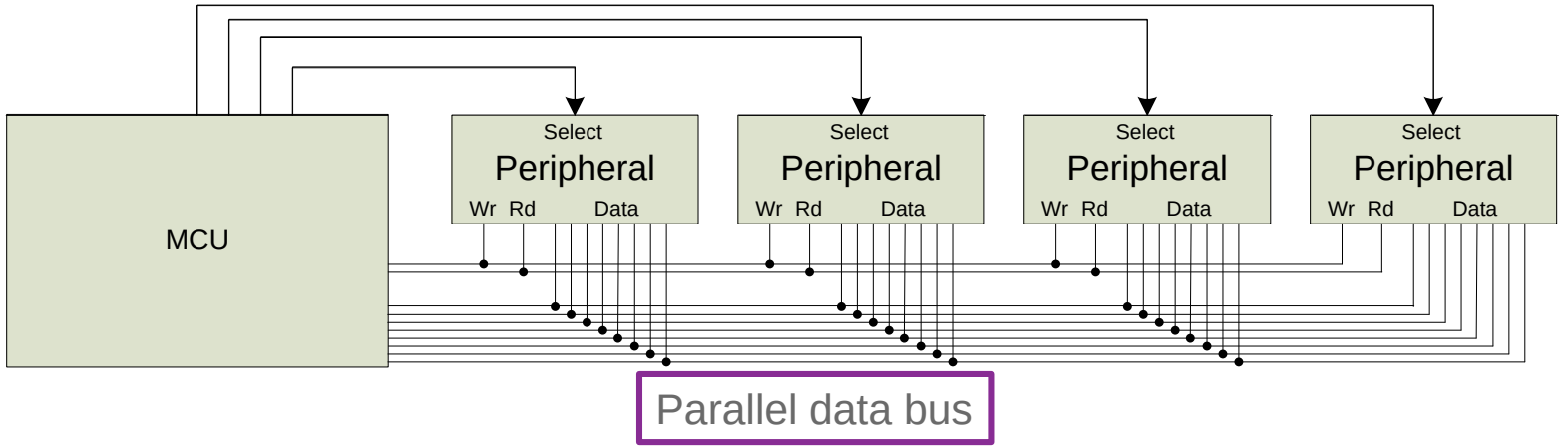
Application
Presentation
Session
Transport
Network
Data Link
Physical

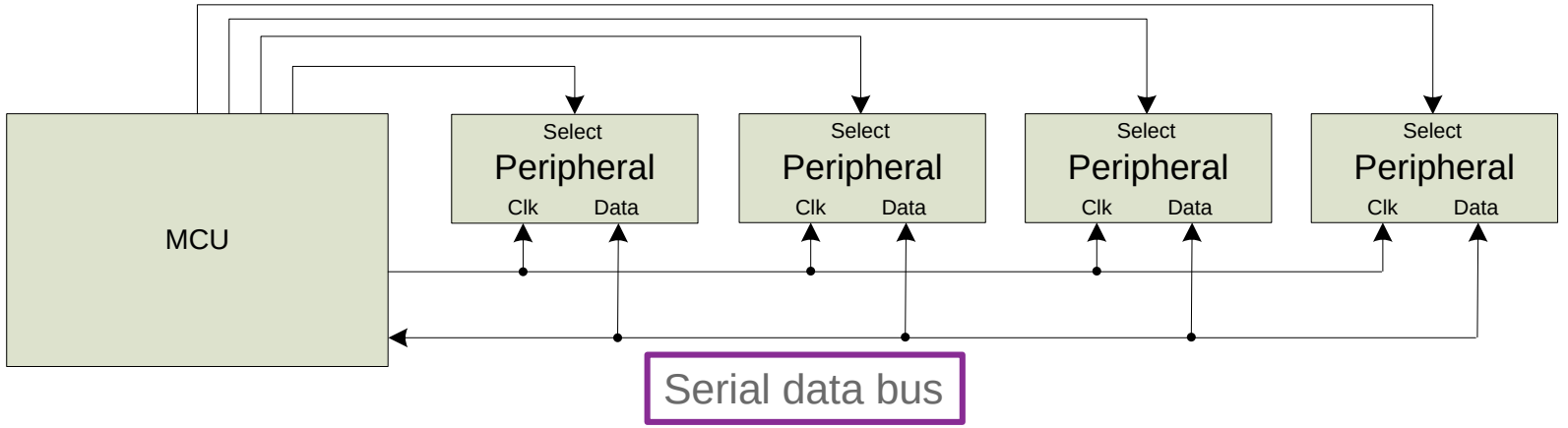
- Helpful to group these rules into layers in a stack
- Example: ***Open System Interconnection (OSI) model***
  1. ***Physical layer:*** Defines how 1s and 0s are represented. Voltage, current, electromagnetic field, light. Amplitude, duration, etc.
  2. ***Data Link layer:*** Has two layers
     - ***Media Access Control layer:*** How nodes share the communication medium. When does a node get to talk on the wire?
     - ***Logical Link Control layer:*** How data is framed how receiver is synchronized (when does the data start?), and how errors are detected
  3. ***Network layer:*** How to route data between nodes, including addressing, handling data too large to fit into one packet, congestion control, and error handling
  4. ***Transport layer:*** How to provide complete, correct data transfer between nodes (called hosts)
  5. ***Session layer:*** How to provide connections between application programs on different nodes
  6. ***Presentation layer:*** Translates data (e.g. encryption and decryption)
  7. ***Application layer:*** Consists of application programs
- OSI model defined for large networks of computing systems (e.g. Internet), not targeted to embedded systems
- Protocols for embedded systems often merge or omit layers/features if not needed

Communication Basics

Start with the foundation – the physical layer

- Communication systems usually ***serialize*** data
  - Don't send all the bits at once
- Why?
  - Reduce number of data signals needed (1, 2, or 4 vs. 8, 32)
  - Reduces package or connector size, weight
  - Simplifies circuit design



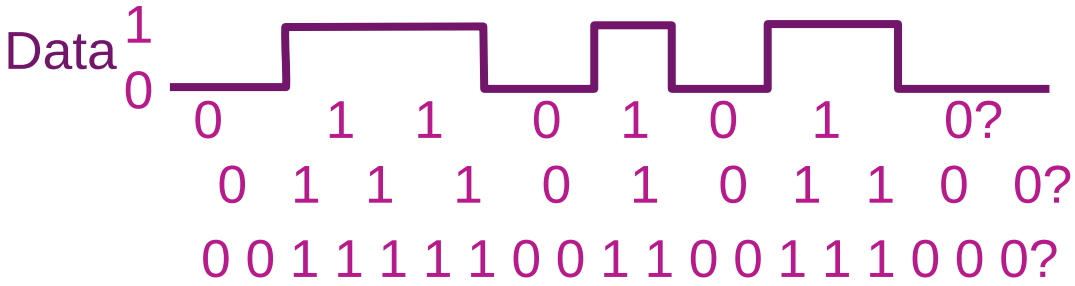


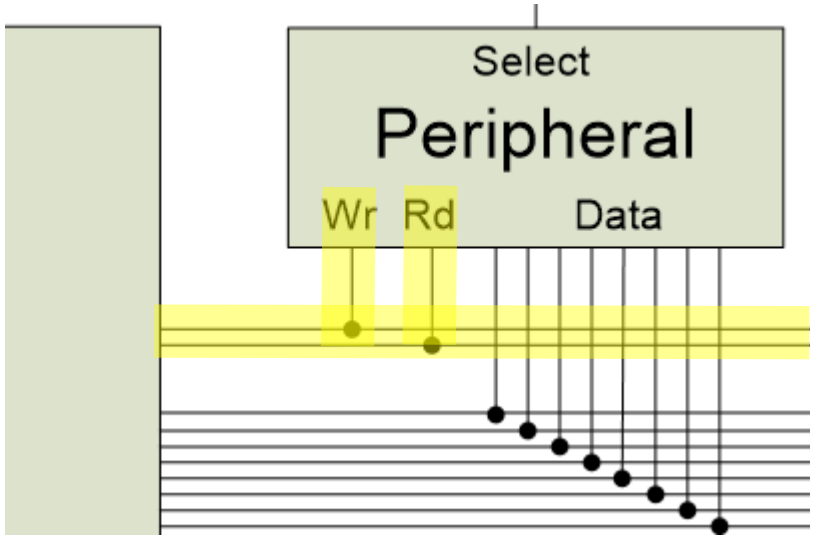
When is the Data Valid?

When should receiver sample the data? Data link layer

- Two approaches
  - **Synchronous**: Transmitter sends a control signal to tell receiver when to sample data
  - **Asynchronous**: Receiver has to determine when data is valid
- **Parallel** communications typically use synchronous communication
  - Already have multiple signals for data (8, 16, 32)
  - Can usually afford an extra signal or two (e.g. Write Enable (Wr), Read Enable(Rd))
- **Serial** communications
  - Goal is to reduce number of signals, so synchronous is less attractive because of extra signals

What does this bit stream mean?

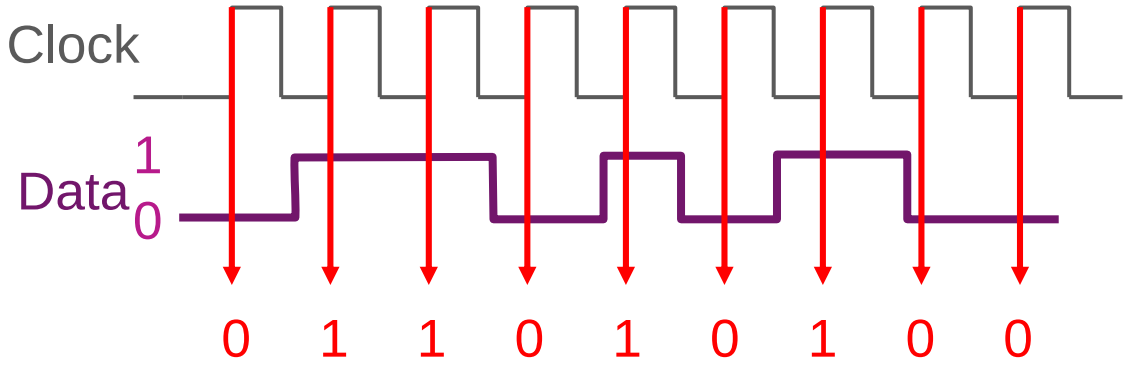




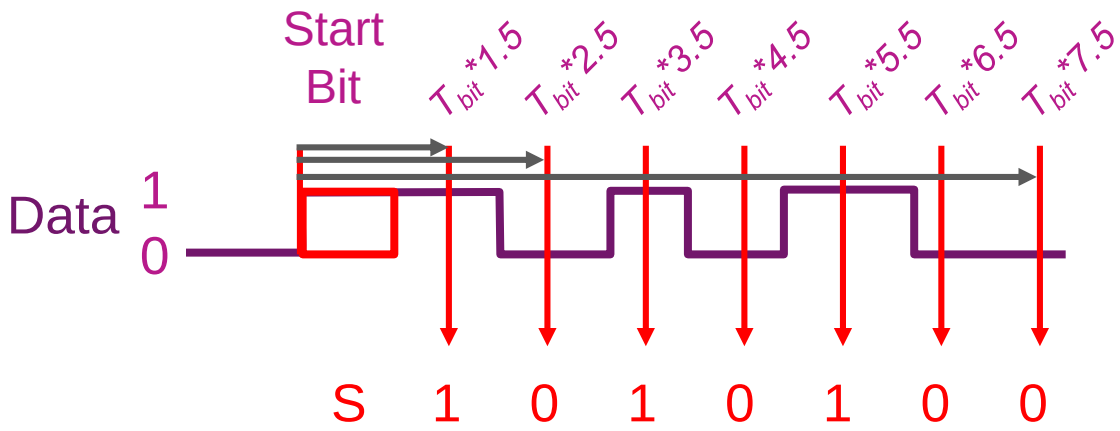
Sampling Data

- Synchronous
  - Use separate clock signal to define bit times
  - Example: Sample data on clock's rising edge

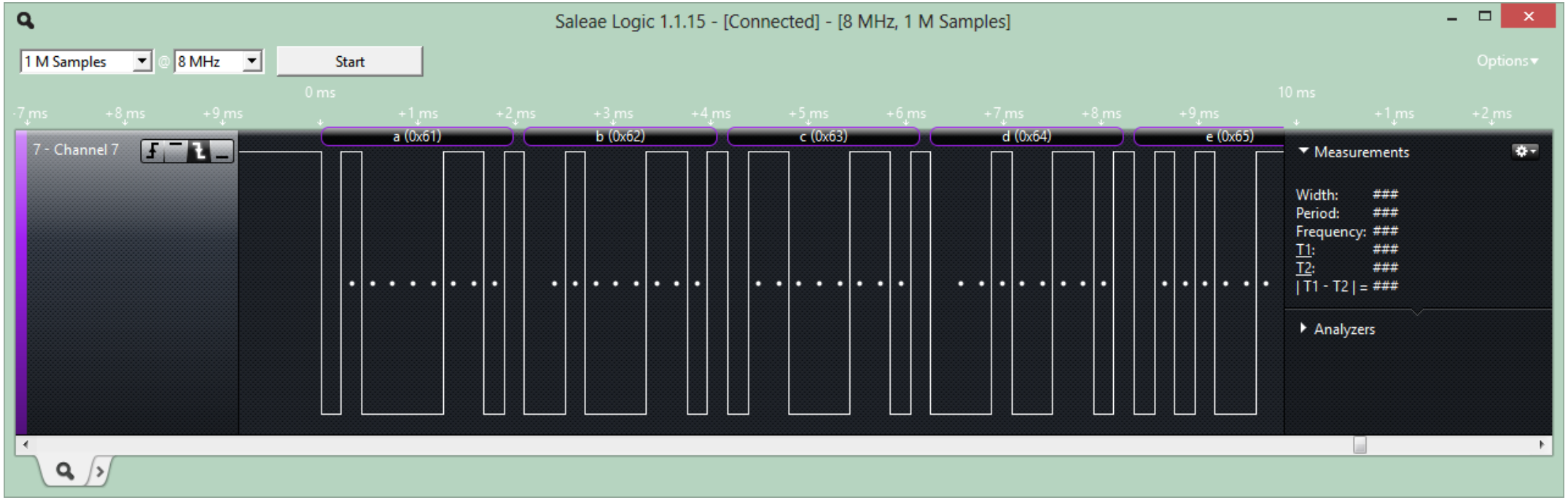
What does this bit stream mean?



- Asynchronous
  - Infer bit times based on ***fixed delays*** from ***reference event***
  - Example
    - Reference event is leading edge of start bit (0 to 1 transition)
    - Sample input data at n+1/2 bit times after beginning of start bit



Tools for Serial Communications Development



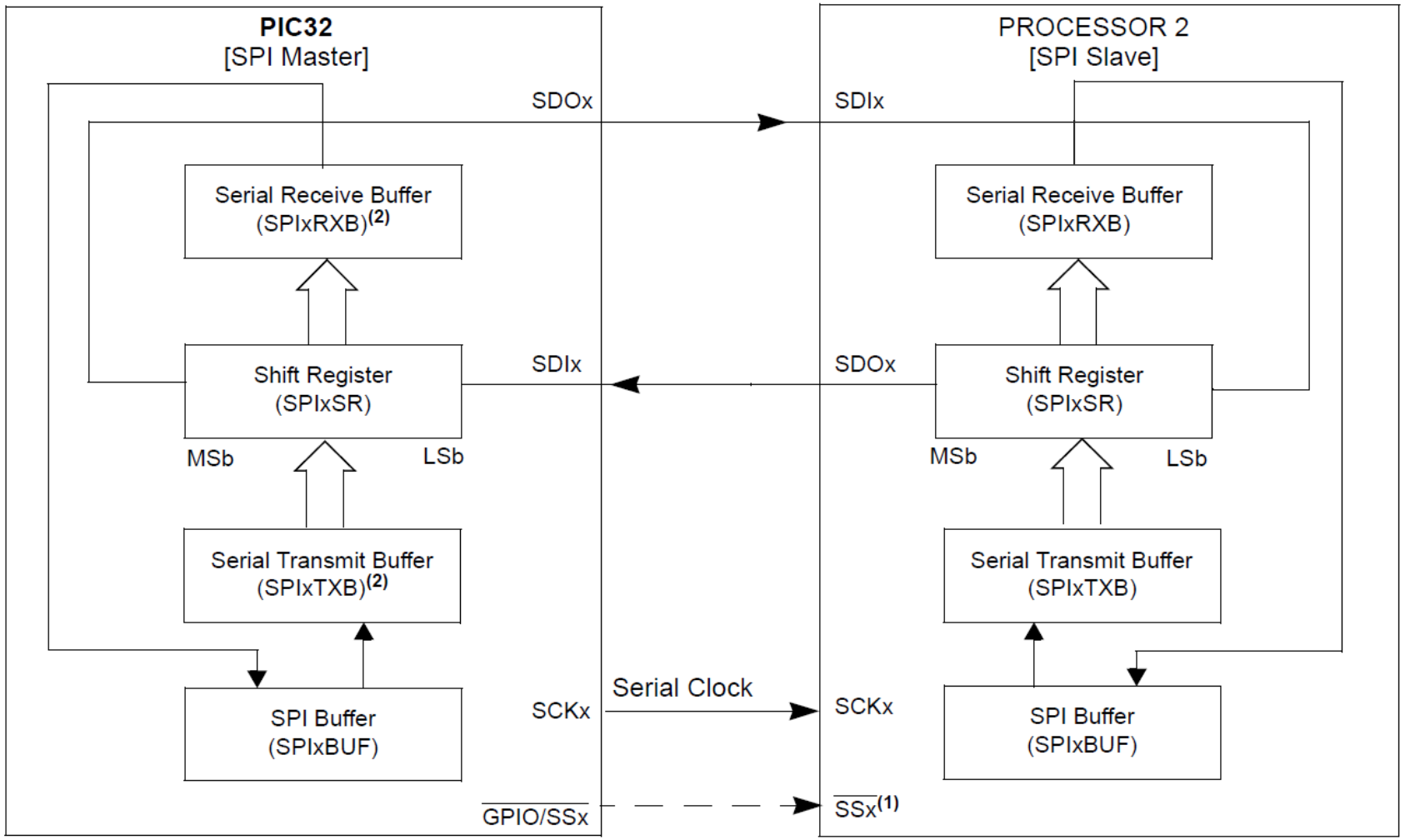
- Tedious and slow to debug serial protocols with just an oscilloscope
- Instead use a logic analyzer to decode bus traffic
- Worth its weight in gold!

- Saelae 8-Channel Logic Analyzer
  - $150 (www.saelae.com)
  - Plugs into PC's USB port
  - Decodes SPI, asynchronous serial, I^2C, 1-Wire, CAN, etc.

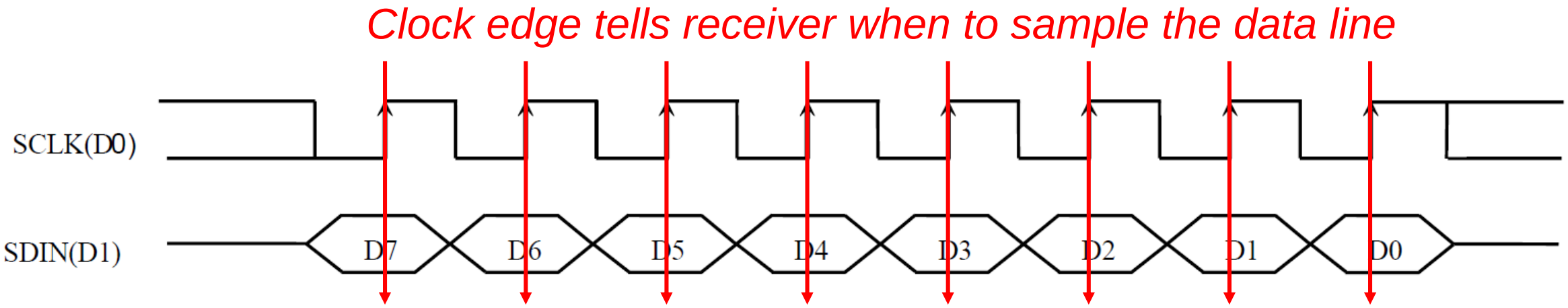
Section 2:
Serial Peripheral Interconnect (SPI)

SPI Basics

- "Ring of shift registers" which exchange data
- Master device generates clock signal (SCKx) which…
  - Shifts data from master to slave one bit at a time
  - Shifts data from slave to master one bit at a time
- Optional Slave Select signal (SSx)
  - Used to identify which slave is being accessed
- SPI defines parts of physical and data link layers



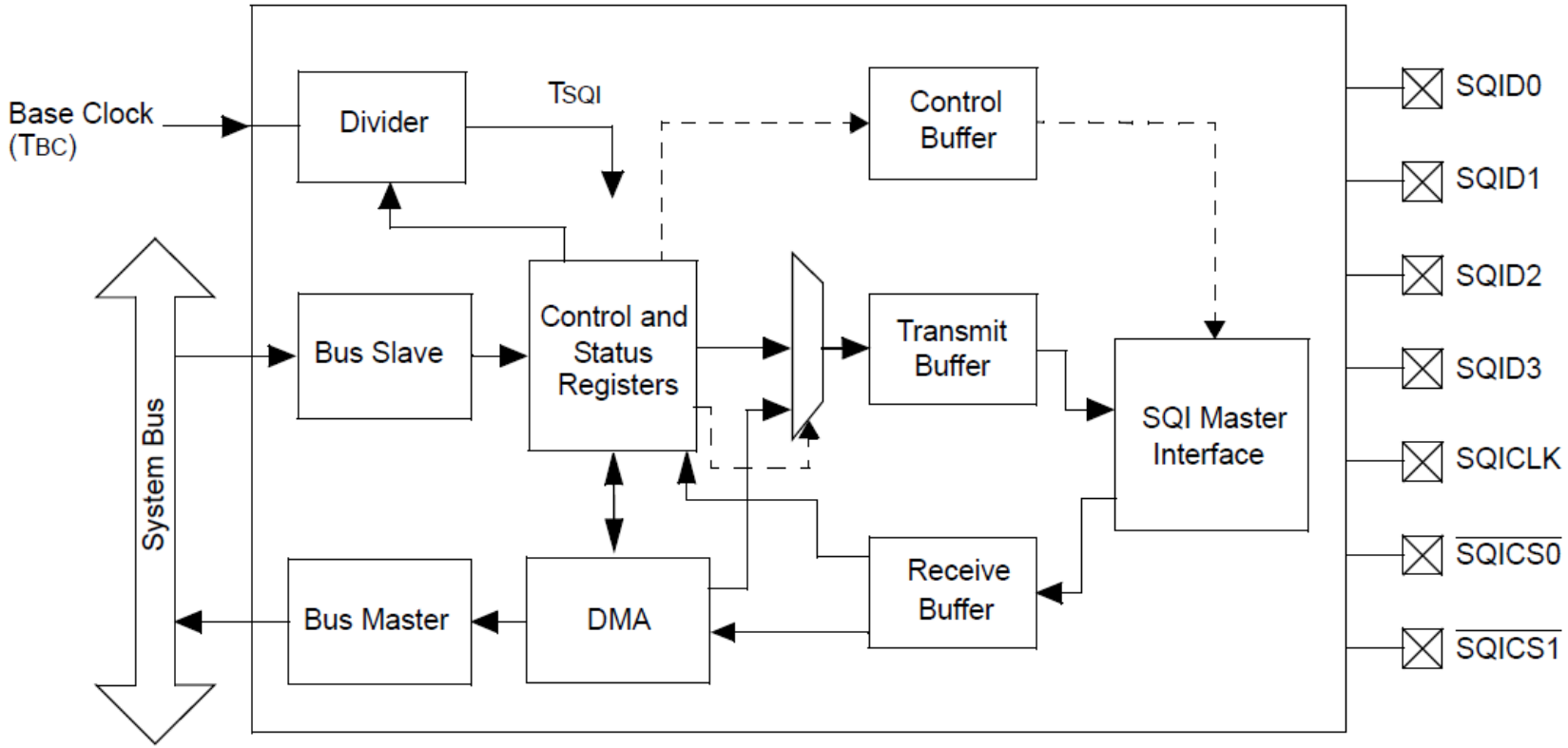
SPI Data Transmission



SPI Data Format for OLED Controller (SSD1306, Solomon Systech)

- Clock signal SCKx
  - Generated by master
  - Defines communication timing
  - SCKx generated by SPI module's baud rate generator by dividing down a reference clock
- Data signal
  - Generated by master on data output pin SDOx
  - D7 (most-significant bit) sent first
  - Sampled by slave when clock rises*
    - * Other versions of SPI use falling clock edge

SQI – Serial Quad Interface

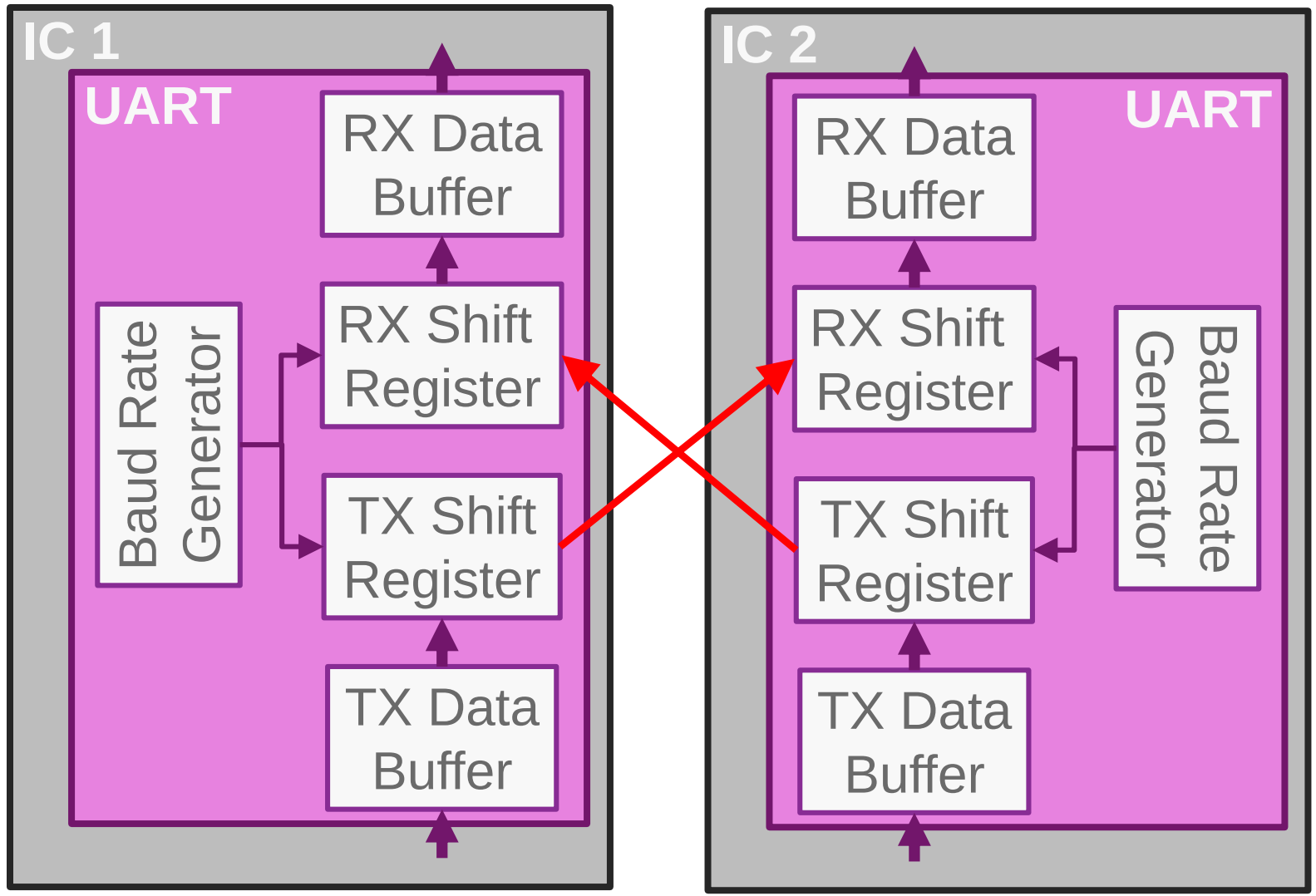


- Similar to SPI but supports wider data (2, 4 bits), so is faster
- Used for communications with Flash, EEPROM storage devices
- Can be configured to operate as SPI (single data lane)

Asynchronous Communication

Asynchronous Serial Basics

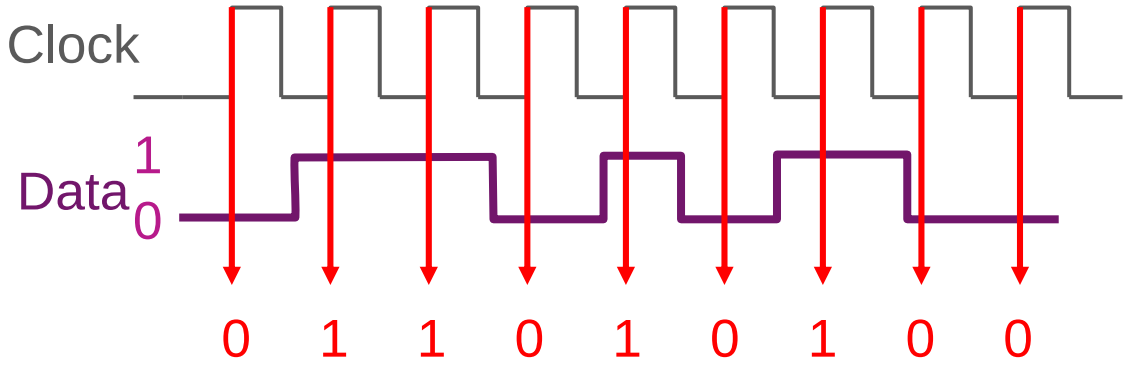
- Similar to SPI, but no external clock signal used
- Peripheral is called a ***UART***
  - Universal = configurable
  - Asynchronous = no clock signal used for communication
  - Receiver/Transmitter = contains both receiver and transmitter
- Defines parts of physical and data link layers



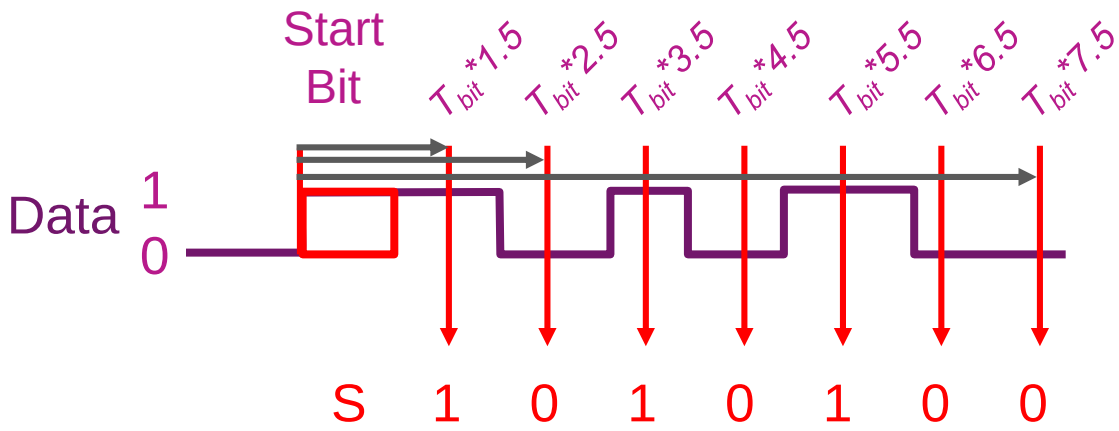
Reminder: Sampling Data

- Synchronous
  - Use separate clock signal to define bit times
  - Example: Sample data on clock's rising edge

What does this bit stream mean?

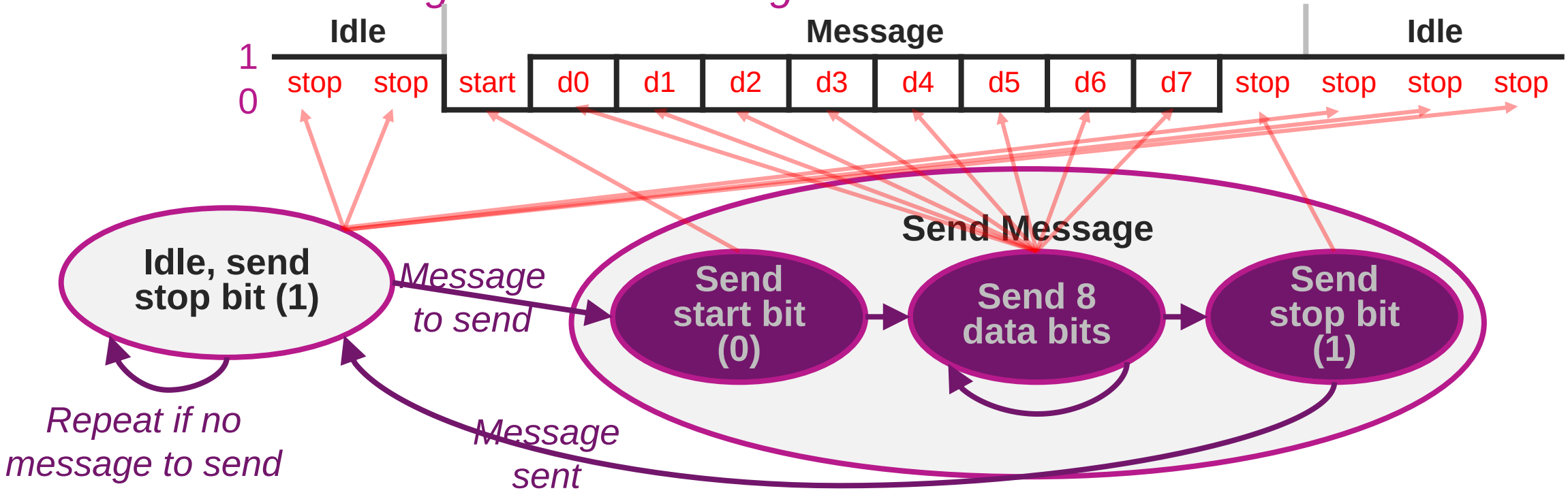


- Asynchronous
  - Infer bit times based on ***fixed delays*** from ***reference event***
  - Example
    - Reference event is leading edge of start bit (0 to 1 transition)
    - Sample input data at n+1/2 bit times after beginning of start bit



Framing

Transmitter inserts framing information to signal data start and end



- Transmitter in Idle state:
  - Send another stop bit (1)
- Transmitter in Send Message state
  - Send Start bit (0),
  - Send data bits, starting with LSB
  - Send Stop bit (1)

Major Asynchronous Communication Options

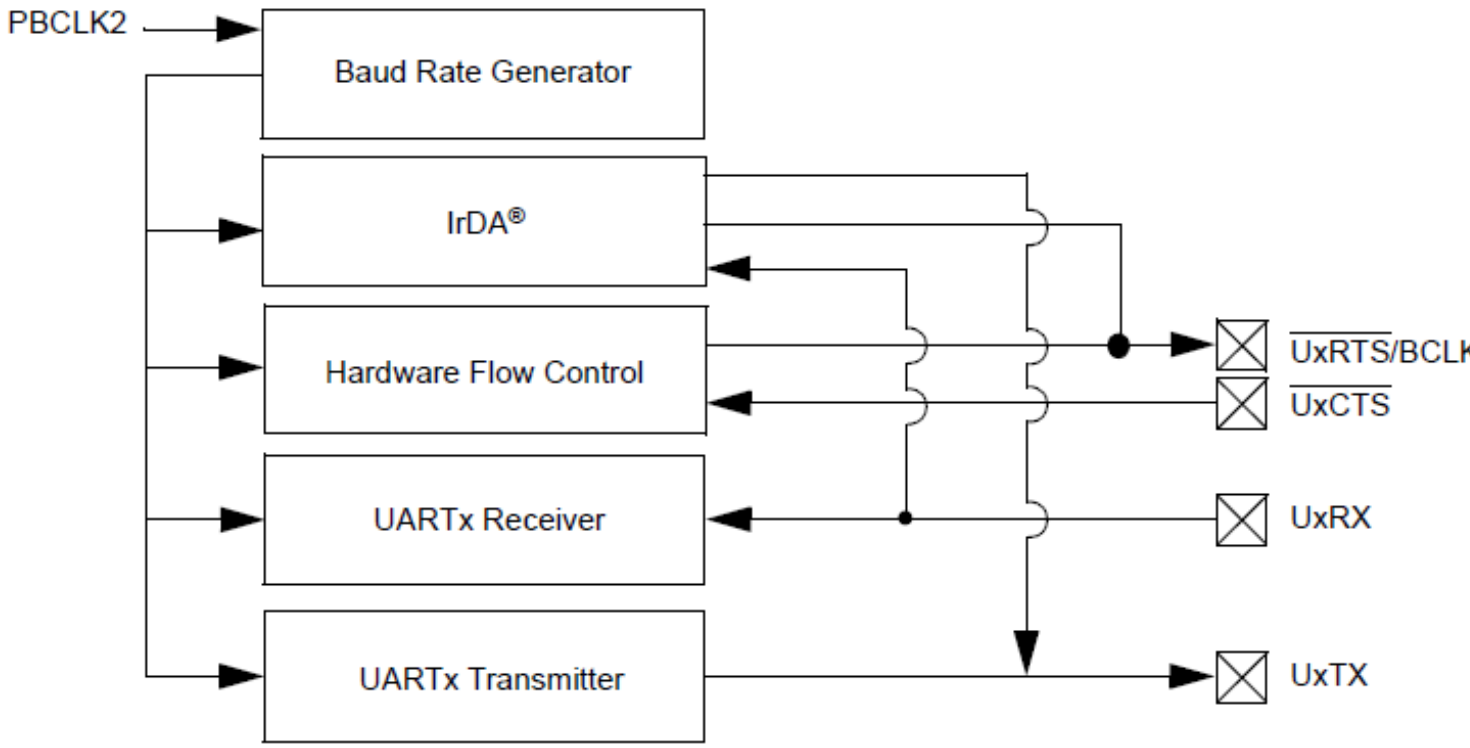
- May have 1 or 2 stop bits
- May have 7, 8 or 9 data bits

- Transmitter may add **parity bit** for error detection
  - Using Even Parity? Set parity bit to make total number of 1s **even**.
  - Using Odd Parity? Make total number of 1s **odd**.
    - Receiver calculates parity based on received data bits and parity bit (not start or stop bits)
    - If parity doesn't match specification (even or odd), then signal an error
  - Can detect an odd number of bit errors, but not an even number

Data		# of 1's	Parity bit for…	
Hex	Binary		Even Parity	Odd Parity
00	0000 0000	0	0	1
3f	0111 1111	7	1	0
a5	1010 0101	4	0	1
16	0001 0110	3	1	0

UART Modules on PIC32MZ EF

- PIC32MZ EF contains six UART modules, U1-U6
- Each module contains
  - Baud rate generator for setting communication speed
  - IrDA support for serial infrared communication
  - Hardware flow control to let one UART ask another UART to pause transmitting
  - Receiver
  - Transmitter



On the Xilinx boards

Check the uartlite datasheet

Polled Data Transmission

UART4_putc Function Sends One Character

- Problem: UART runs much slower than CPU!
  - CPU: up to 200 million instructions per second
  - UART: maybe 20 thousand bytes per second
- Might still be sending previous character when we want to send next one
- Need to avoid overwriting data in TX data buffer
- Must wait until TX data buffer has space available (is not full) before writing to it

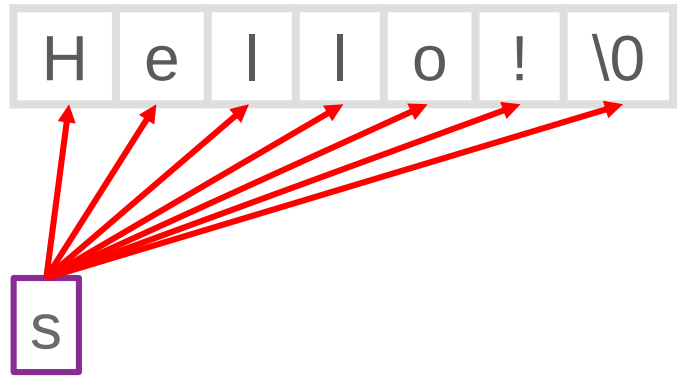
```
void UART4_putc (char c)
{
    while (U4STAbits.UTXBF)
        ;  // wait until transmit buffer not full
    U4TXREG = c;  // transmit character
}
```

Polled Data Transmission – String of Characters

UART4_puts Function Sends a String

- Input s is pointer to the first character of the string (H)
- Use a loop to send all characters
  - End of string marked by null character (\0 value)
  - *s != '\0': Check to see if s is pointing to \0
    - if so, then skip loop body
    - if not, execute loop body and try to repeat
  - Use pointer s to step through all characters in string
    - s++: ++ advances s to point to next character
- Loop stops when s reaches end of string \0

```
void UART4_puts (char *s)
{
    while (*s != '\0')
        UART4_putc (*s++);
}
```



Polled Data Reception

UART4_getc Function Reads One Character

- Reading the received data buffer before the data has arrived gives us old data
- Still have to worry about speed difference between UART and CPU

```
char UART4_getc(void) {
    while (!U4STAbits.URXDA)
        ; // wait until character received
    return U4RXREG; // read character
}
```

- Solution: wait for status flag URXDA to become 1
  - UART sets URXDA to 1 when new data arrives
- Then read UxRXREG to get that data
  - Reading UxRXREG makes UART update the URXDA flag, clearing it to 0 if no more data is in RX data buffer
  - RX data buffer has space for eight elements

Section 7: Advanced Communication Concepts and Example Protocols

Review: Protocol Stack Concepts

All nodes must follow the same or compatible rules

Application
Presentation
Session
Transport
Network
Data Link
Physical

- Helpful to group these rules into layers in a stack
- Example: ***Open System Interconnection (OSI) model***
  1. ***Physical layer:*** Defines how 1s and 0s are represented. Voltage, current, electromagnetic field, light. Amplitude, duration, etc.
  2. ***Data Link layer:*** Has two layers
     - ***Media Access Control layer:*** How nodes share the communication medium. When does a node get to talk on the wire?
     - ***Logical Link Control layer:*** How data is framed how receiver is synchronized (when does the data start?), and how errors are detected
  3. ***Network layer:*** How to route data between nodes, including addressing, handling data too large to fit into one packet, congestion control, and error handling
  4. ***Transport layer:*** How to provide complete, correct data transfer between nodes (called hosts)
  5. ***Session layer:*** How to provide connections between application programs on different nodes
  6. ***Presentation layer:*** Translates data (e.g. encryption and decryption)
  7. ***Application layer:*** Consists of application programs
- OSI model defined for large networks of computing systems (e.g. Internet), not targeted to embedded systems
- Protocols for embedded systems often merge or omit layers/features if not needed

Key Concepts

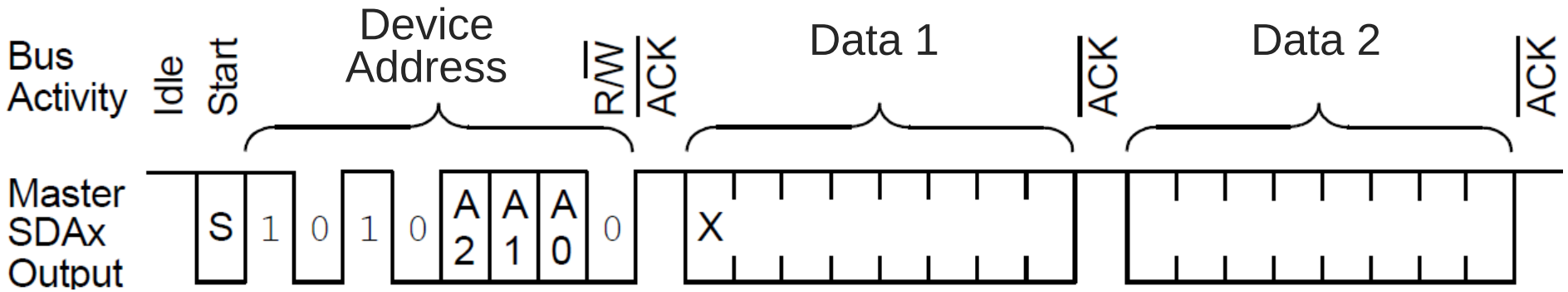
- How do we detect data transmission errors?
- What's in a message besides data?
- How can we make the communication system scale up to large sizes easily?
- How can we increase the communication speed?
- How can we transmit data wirelessly?

How Do We Detect Errors?

- Approach
  - Transmitter sends extra error-detection information along with data
  - Receiver ***recalculates*** the error-detection information based on received data,
  - Receiver ***compares*** recalculated version with received version
  - If these don't match, then the message was corrupted and should be discarded
- Examples:
  - Parity: There is an odd number of ones in this message
  - Checksum: If you add up all the bytes in this message, the sum ends in 0x38
  - CRC: If you process all the bytes in this message this way (e.g. by shifting and exclusive-oring them together), the result ends in 0x68
- Acknowledgements
  - Receiver must acknowledge each message, else transmitter will resend it

What's in a Message Besides Data?

There's more than just data



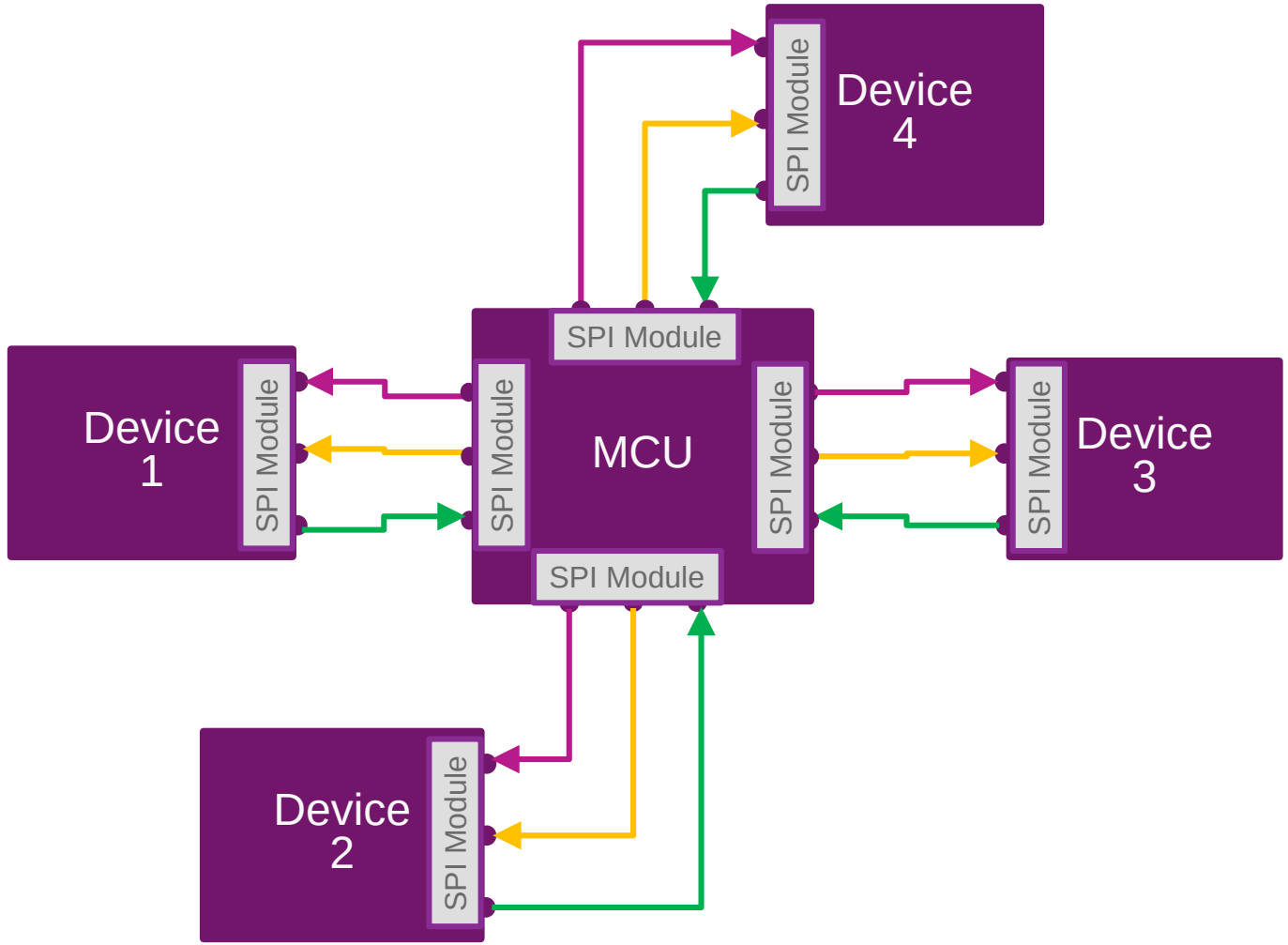
Example of I²C message

- Message holds data and other information
- Data: May multiple bytes per message
- Other information
  - Framing information: When message starts, stops
  - Error detection information: Parity or CRC
  - Acknowledgement: Received correctly?
  - More (discussed soon): Device address, operation (read, write), data length, etc.
- Example: I²C (Inter-integrated circuit bus)
  - Start condition
  - Device address
  - Read or write command
  - Acknowledgement(s) from slave
  - Multiple bytes of data

How Can we Make Scaling Up Easier?

Supporting many communication devices

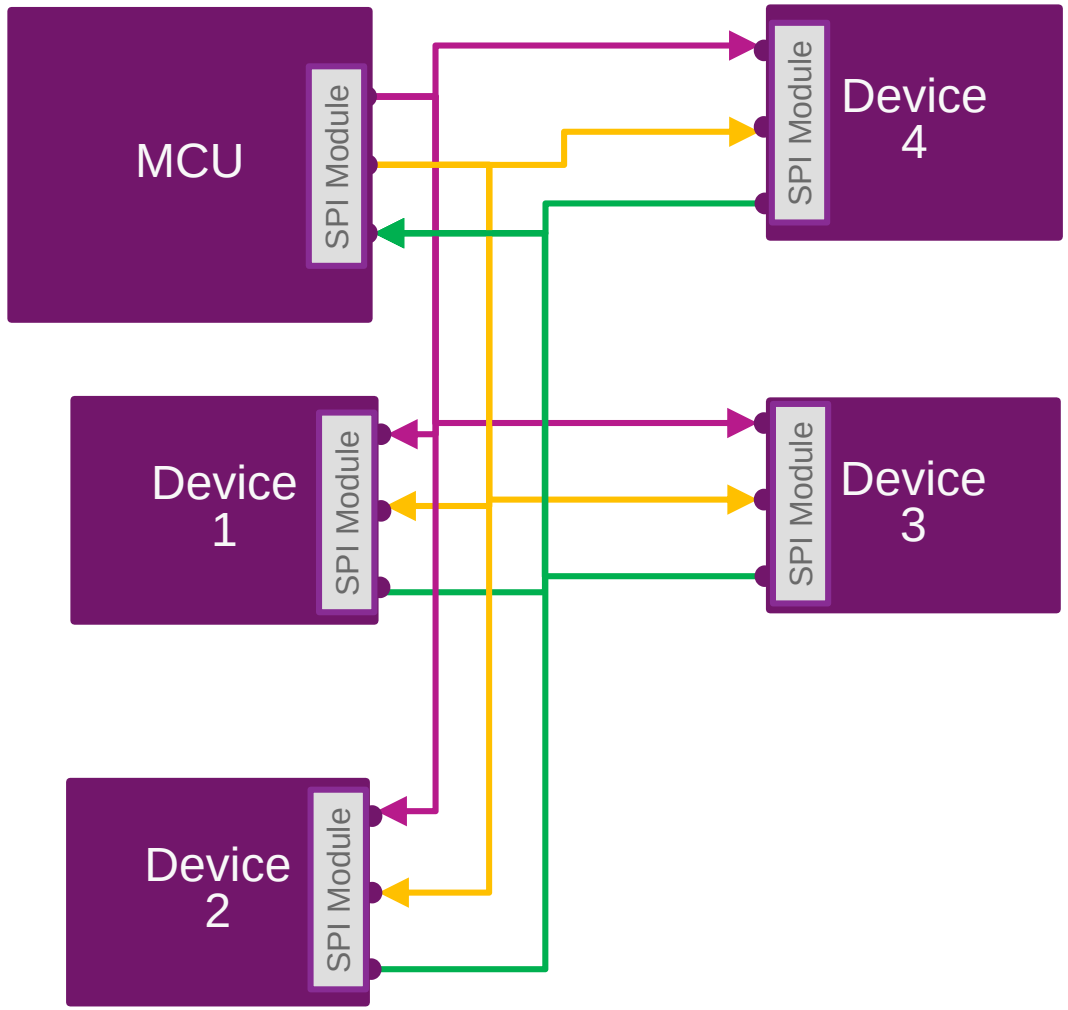
- Dedicated communication links
  - Each pair of communicating devices has a dedicated set of wires and a pair of communication modules
- Problem
  - We need many ports and sets of wires to talk with multiple devices
  - Doesn't work well for systems with many devices
- Instead, can multiple devices ***share*** same the communication signals and wires?



Sharing: Who Gets to Talk When?

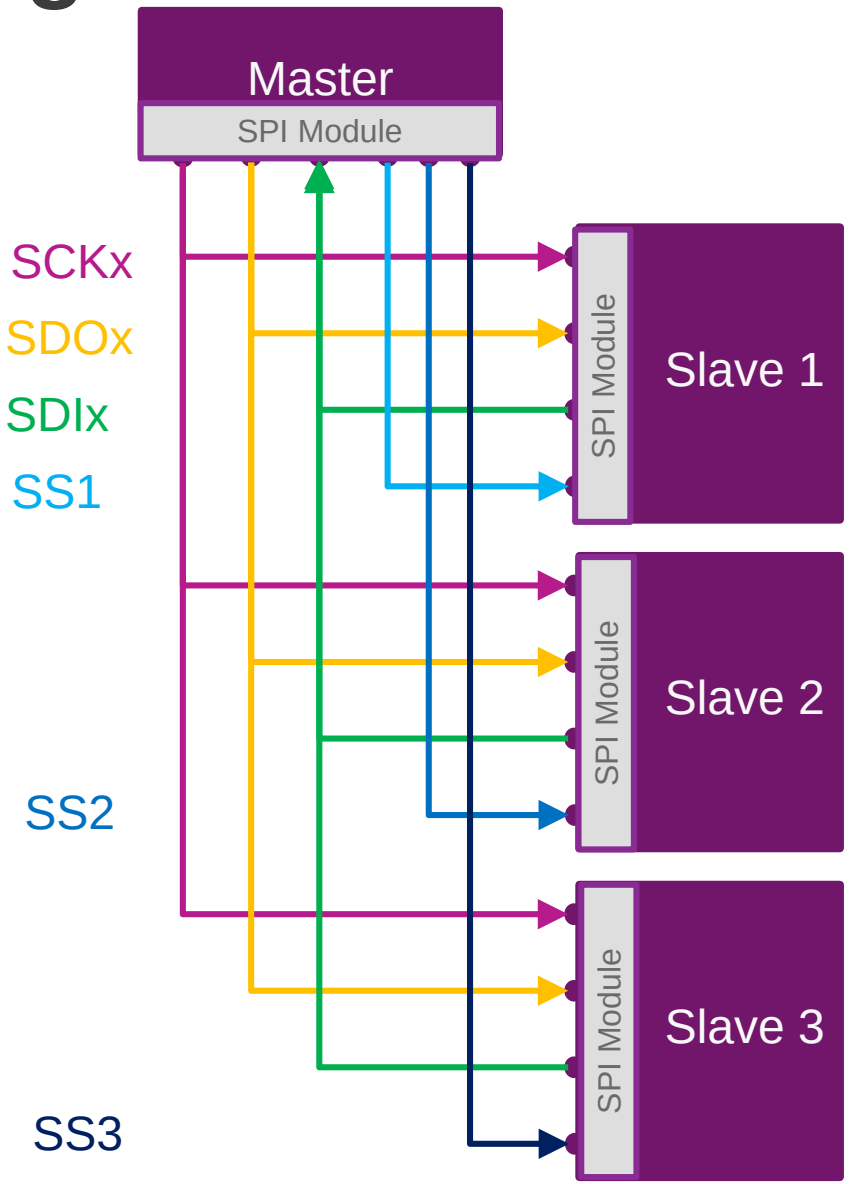
Data link layer and media access control

- Now each node needs ***just one*** communication interface
- All nodes share the bus (communication medium)
- Need a ***media access control*** method
  - Determines which node talks when
- Categories
  - Master/Slave: master tells each node when it can talk
  - Multiple access: no master needed, nodes decide on their own when to talk

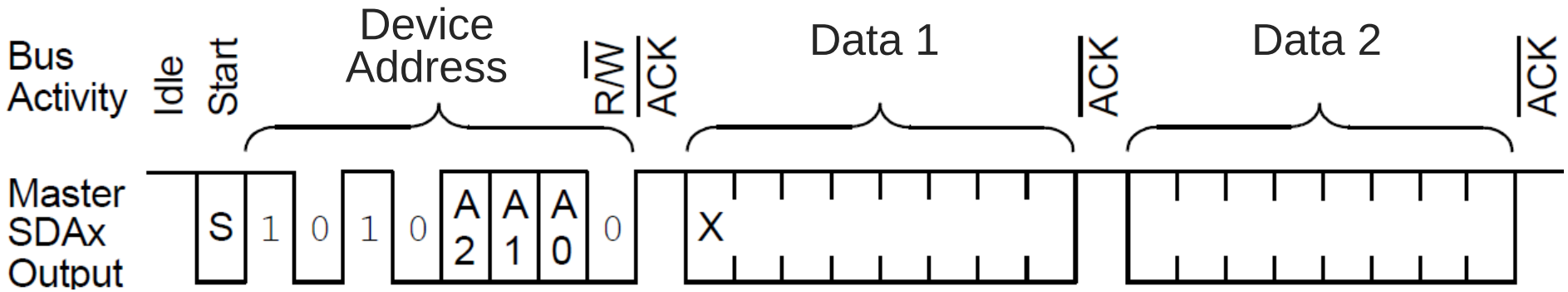


Master/Slave with Select Signals

- Multiple SPI slave devices can share clock and data lines
- Master only needs one SPI module
- Select slave by asserting its slave select line (SS1, SS2, SS3)
- Only one slave select line will be active at a time



Master/Slave with Address in Message



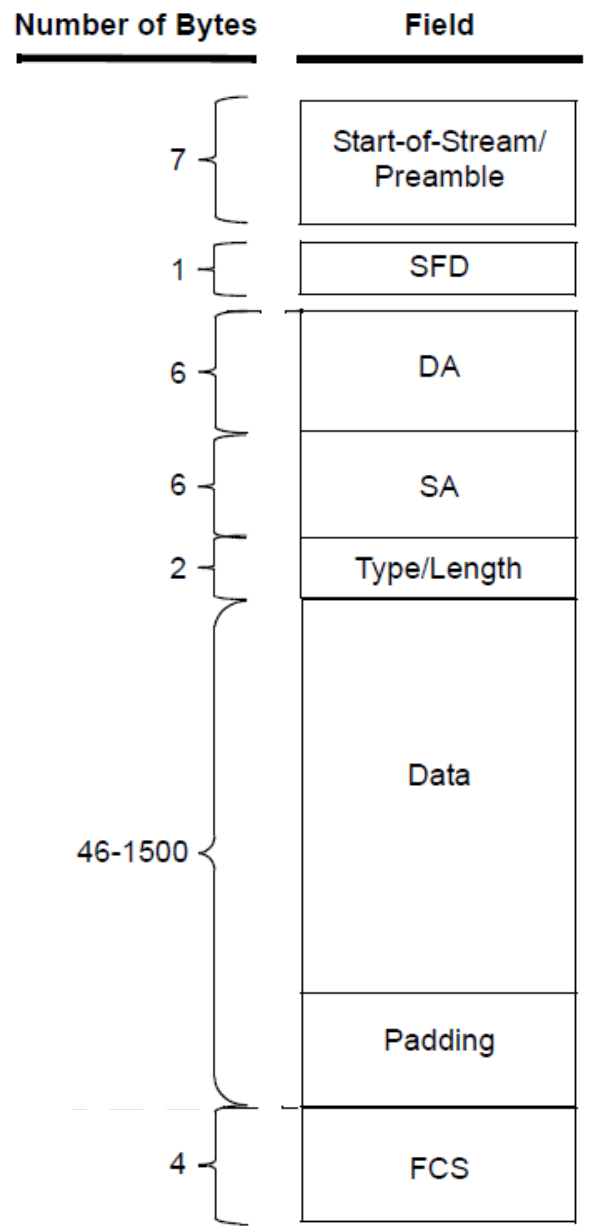
Example of I²C message

- Use data line for both address and data
- Master sends slave device address
- Data is sent by master (if write operation) or slave (if read operation)
- Slave device only processes messages with its own address, ignores other messages
- Typically also have broadcast address to send data to all slaves

Example Protocol: Ethernet

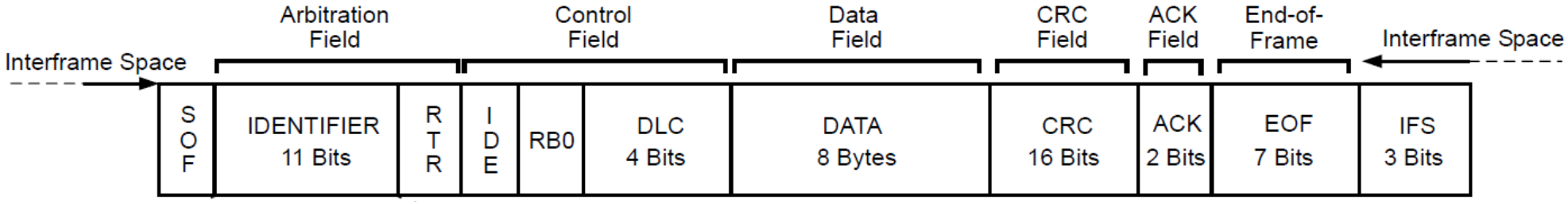
IEEE 802.3

- Key features
  - 10 Mbps/100 Mbps/1 Gbps/10 Gbps data rates
  - 48 bit (6 byte) addresses for source, destination
  - Broadcast, multicast, and unicast message addressing
  - 46 to 1500 bytes of data per message
  - 32-bit CRC (frame check sequence) for error detection
- Improvements
  - Standard Ethernet: Devices share the media (wires) by connecting to a hub.
  - Switched Ethernet: Devices don't share the media. Switch (not hub) has an Ethernet controller and dedicated media for each device.
    - Raises throughput and eliminates collisions



Example Protocol: CAN

Controller Area Network



- Key features
  - Up to 1 Mbps bit rate
  - 11 or 29 bit addresses
  - Up to 8 bytes of data per message
  - 15-bit CRC for error detection
  - ACK field for receiver acknowledgements
  - Remote transmit (read) request
- Easy to analyze responsiveness
  - Highest priority message always wins
  - Given set of all possible messages, can calculate maximum time for any message to get through
- Used in automotive and industrial control networks