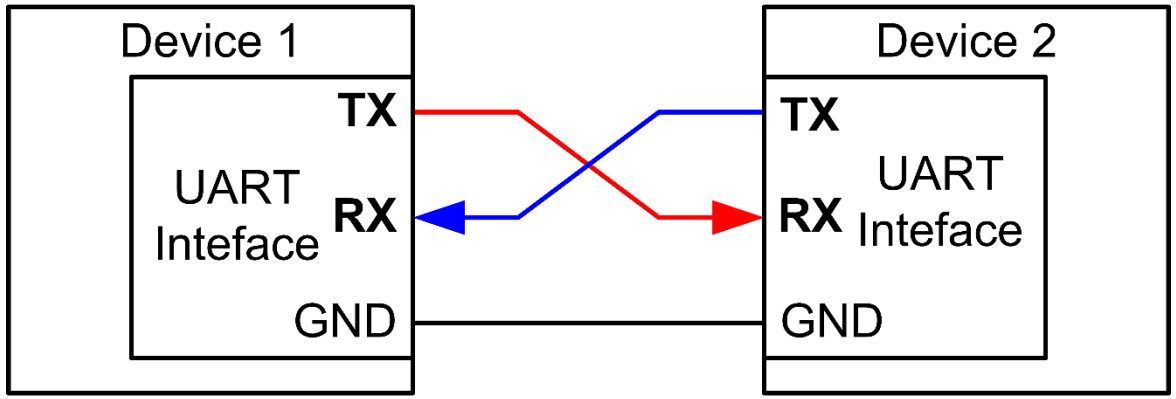
Serial Communication

Suresh C
Assistant Professor EEE DEPT

Universal Asynchronous Receiver and Transmitter (UART)

- Universal
  - UART is programmable.
- Asynchronous
  - Sender provides no clock signal to receivers



UART

- Universal Asynchronous Receiver/Transmitter
- Hardware that translates between parallel and serial forms
- Commonly used in conjunction with communication standards such as EIA, RS-232, RS-422 or RS-485
- The universal designation indicates that the data format and transmission speeds are configurable and that the actual electric signaling levels and methods (such as differential signaling etc.) typically are handled by a special driver circuit external to the UART.

Protocol

- Each character is sent as
  - a logic *low* **start** bit
  - a configurable number of data bits (usually 7 or 8, sometimes 5)
  - an optional parity bit
  - *one or more logic high* **stop** bits
  - with a particular bit timing ("baud")
- Examples
  - "9600-N-8-1" → <baudrate><parity><databits><stopbits>
  - "9600-8-N-1" → <baudrate><databits><parity><stopbits>

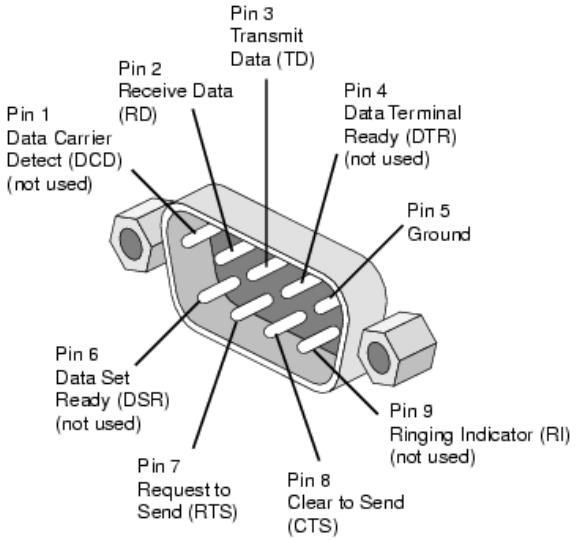
- UART is actually a generic term that includes a large number of different devices/standards.
  - RS-232 is a standard that specifies
    - “electrical characteristics and timing of signals, the meaning of signals, and the physical size and pin out of connectors.

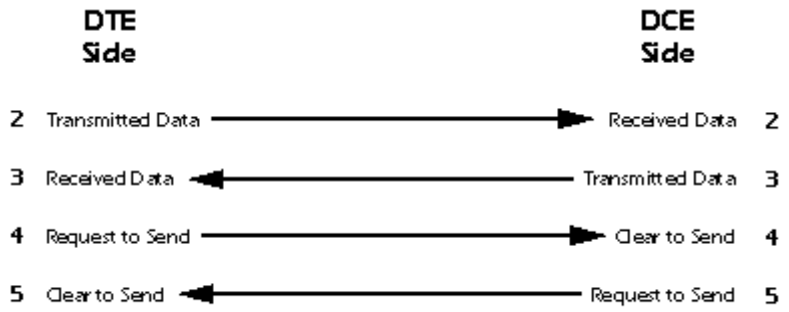
Signals (only most common)

- The **RXD** signal of a UART is the signal receiving the data. This will be an input and is usually connected to the TXD line of the downstream device.
- The **TXD** signal of a UART is the signal transmitting the data. This will be an output and is usually connected to the RXD line of the downstream device.
- The **RTS#** (Ready to Send) signal of a UART is used to indicate to the downstream device that the device is ready to receive data. This will be an output and is usually connected to the CTS# line of the downstream device.
- The **CTS#** (Clear to Send) signal of a UART is used by the downstream device to identify that it is OK to transmit data to the upsteam device. This will be an input and is usually connected to the RTS# line of the upstream device.

DB9 stuff

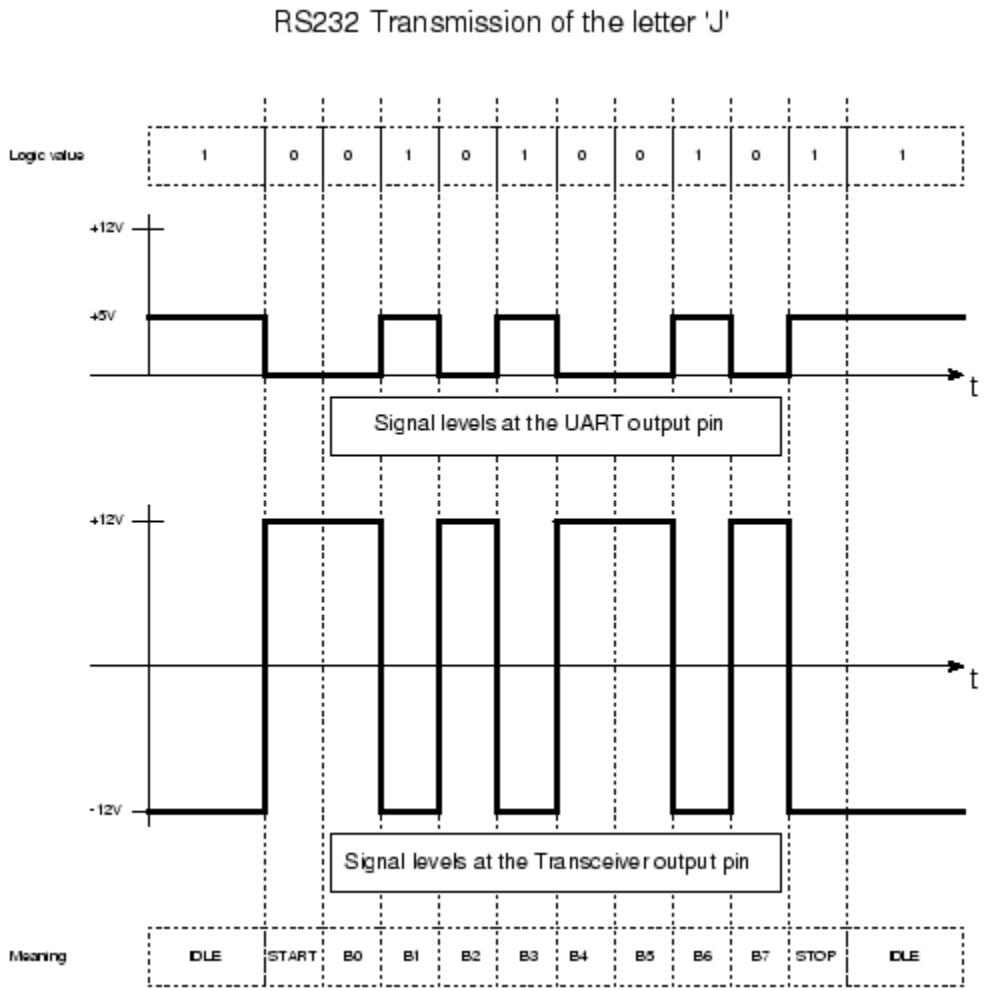
- DTE vs DCE
- Pinout of a DCE?
- Common ground?
- Noise effects?





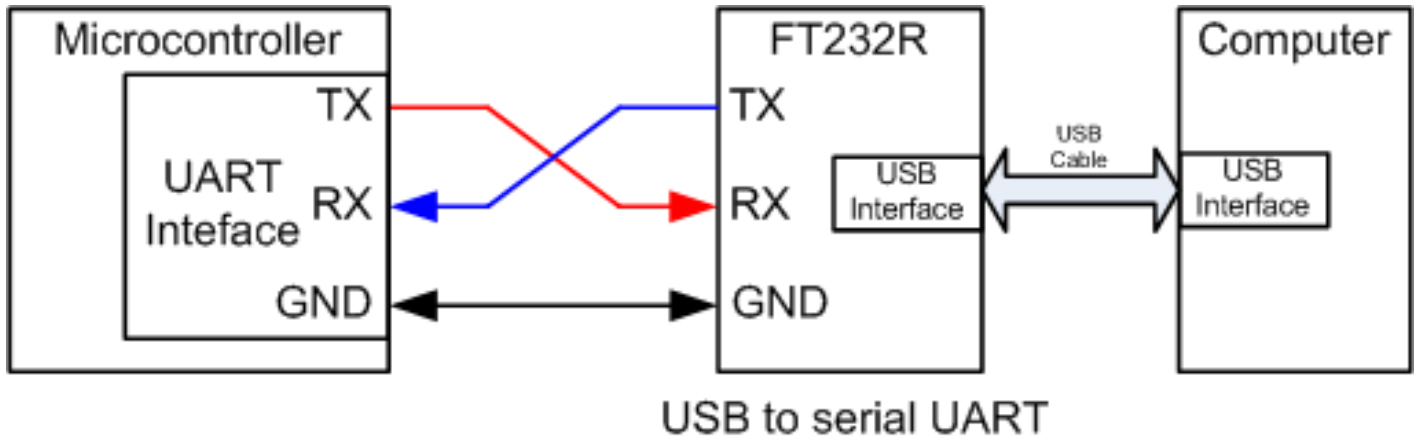
Pin Number	Signal	Description
1	DCD	Data carrier detect
2	RxD	Receive Data
3	TxD	Transmit Data
4	DTR	Data terminal ready
5	GND	Signal ground
6	DSR	Data set ready
7	RTS	Ready to send
8	CTS	Clear to send
9	RI	Ring Indicator

RS-232 transmission example



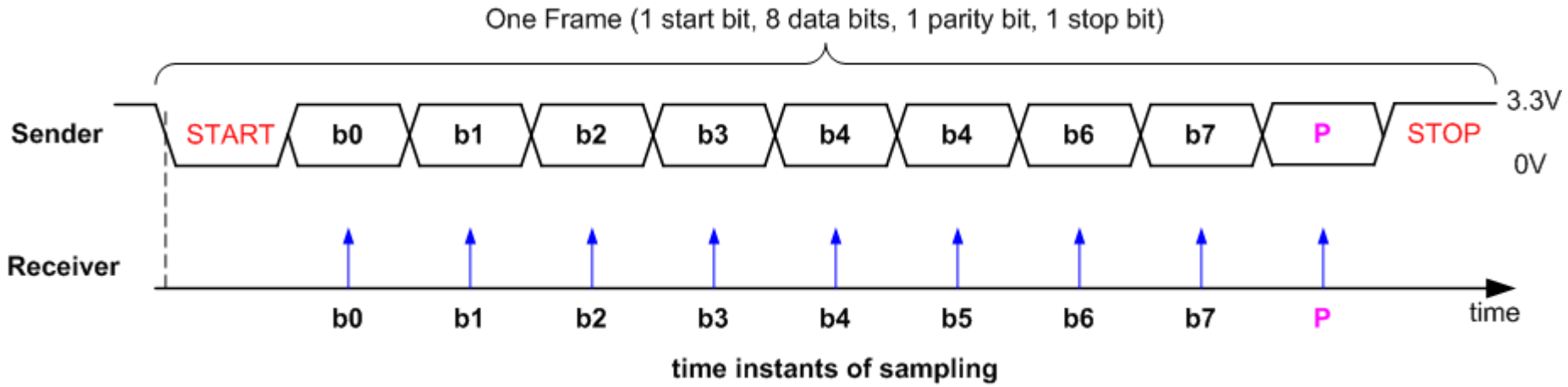
Connecting to PC

- FT232R converts the UART port to a standard USB interface



USB to serial UART

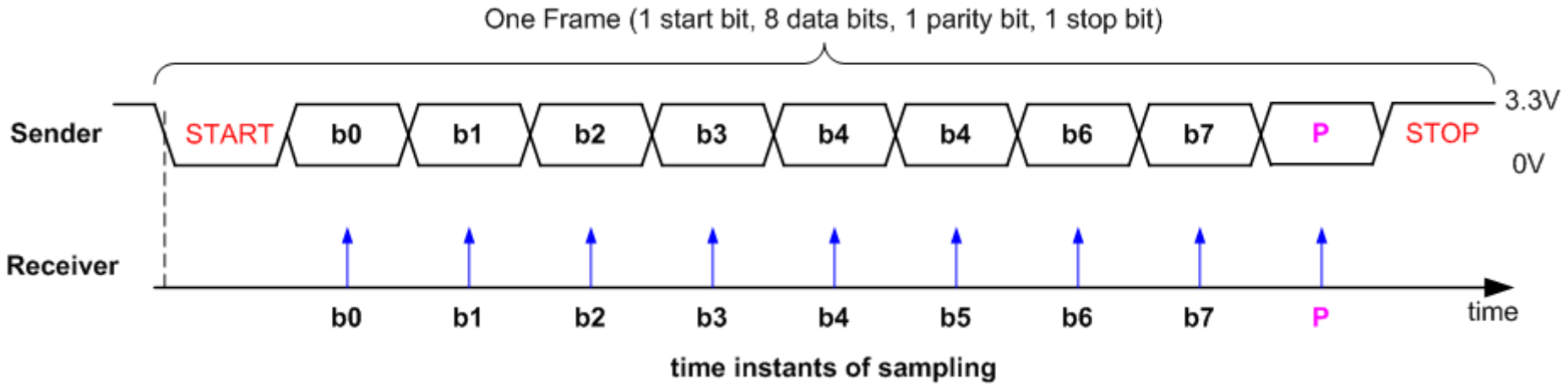
Data Frame



Tolerate 10% clock shift during transmission

- Data frame
  - One logic-low start bit
  - Data (LSB first or MSB first, and size of 7, 8, 9 bits)
  - One optional parity bit
  - One or two logic-high stop bits

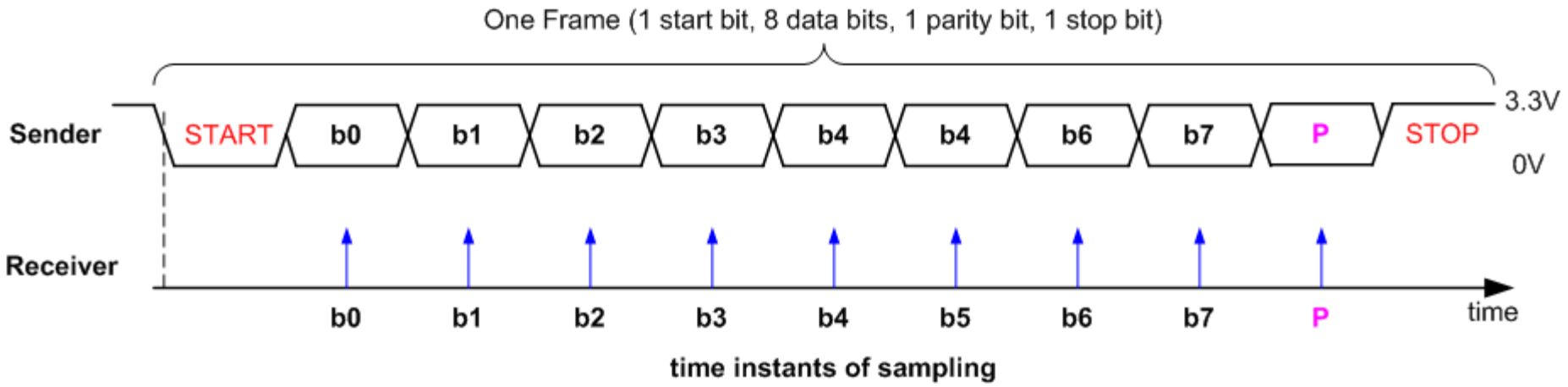
Data Frame



Tolerate 10% clock shift during transmission

- Receiver must know the transmission rate
- Receiver uses a timer to time when it should sample
- To tolerate clock shift, receiver only samples at the middle of each bit
- The start bit inform receiver to reset its timer
- Receiver is only synchronized with the sender on every start bit.
  - Only needs to be accurate enough to read up to 9 bits

Data Frame



Tolerate 10% clock shift during transmission

- Receiver reports "frame error" if the stop bit is not detected
- Another start bit can appear immediately after stop bit
  - Receiver then resynchronizes with sender on the new start bit

Data Frame


9-bit word Length, 1 stop bit
Start bit
Data Frame
Bit0
Bit1
Bit 2
Bit 3
Bit4
Bit5
Bit6
Bit7
Bit8
Optional Parity Bit
Stop bit
Next Start bit
Next D
8-bit word Length, 1 stop bit
Start bit
Data Frame
Bit0
Bit1
Bit 2
Bit 3
Bit4
Bit5
Bit6
Bit7
Optional Parity Bit
Stop bit
Next Start bit
Next Data Fra

Baud Rate

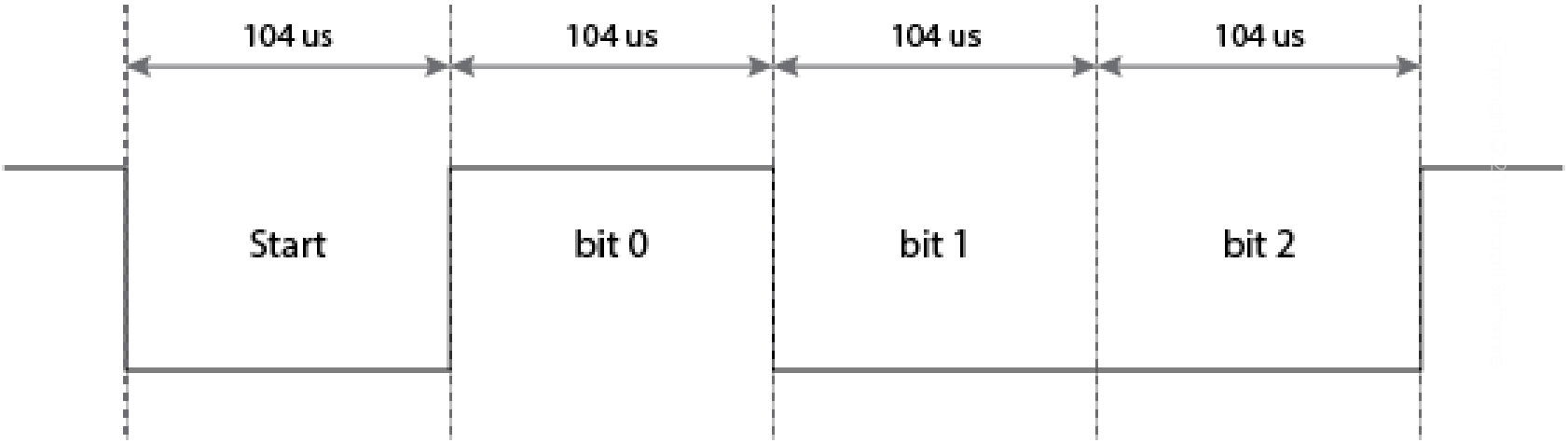
- Historically used in telecommunication to represent the number of pulses physically transferred per second
- In digital communication, baud rate is the number of bits physically transferred per second
- Example:
  - Baud rate is 9600
  - each frame: a start bit, 8 data bits, a stop bit, and no parity bit.
  - Transmission rate of actual data

    9600/8 = 1200 bytes/second

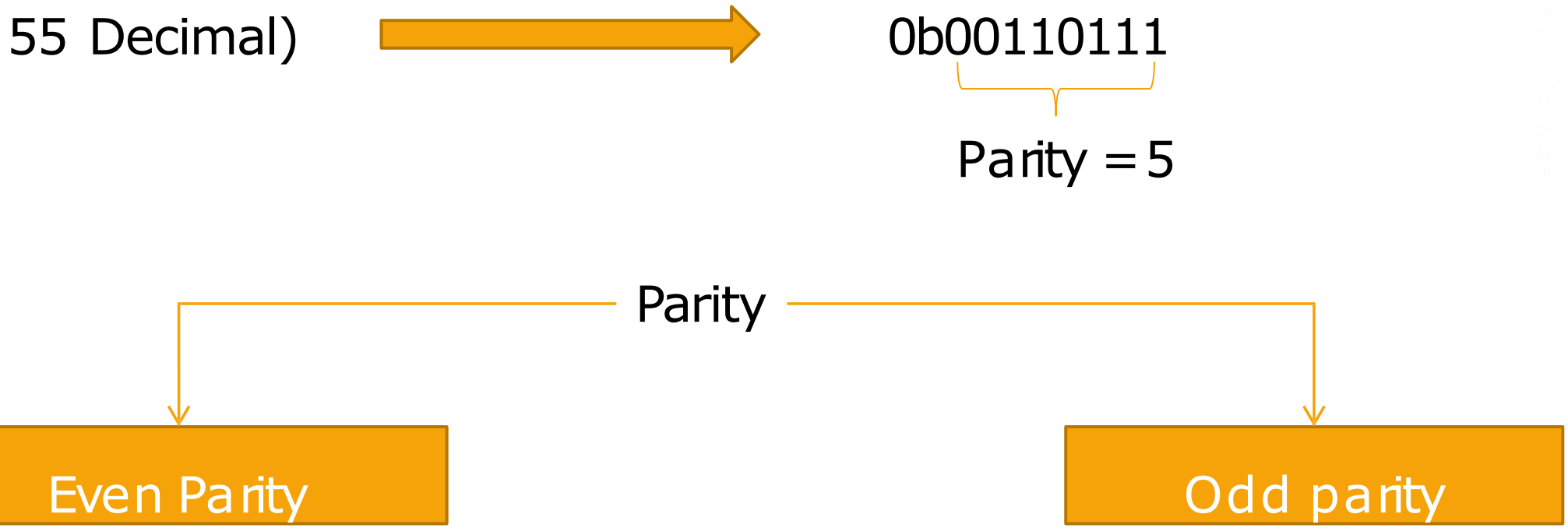
    9600/(1 + 8 + 1) = 960 bytes/second
  - The start and stop bits are the protocol overhead

- The significance of baud rate is **how fast the** data is sent over a serial line.
- It"s usually expressed in units of bits-per-second (bps). If you invert the baud rate, you can find out, just how long it takes to transmit a single bit.

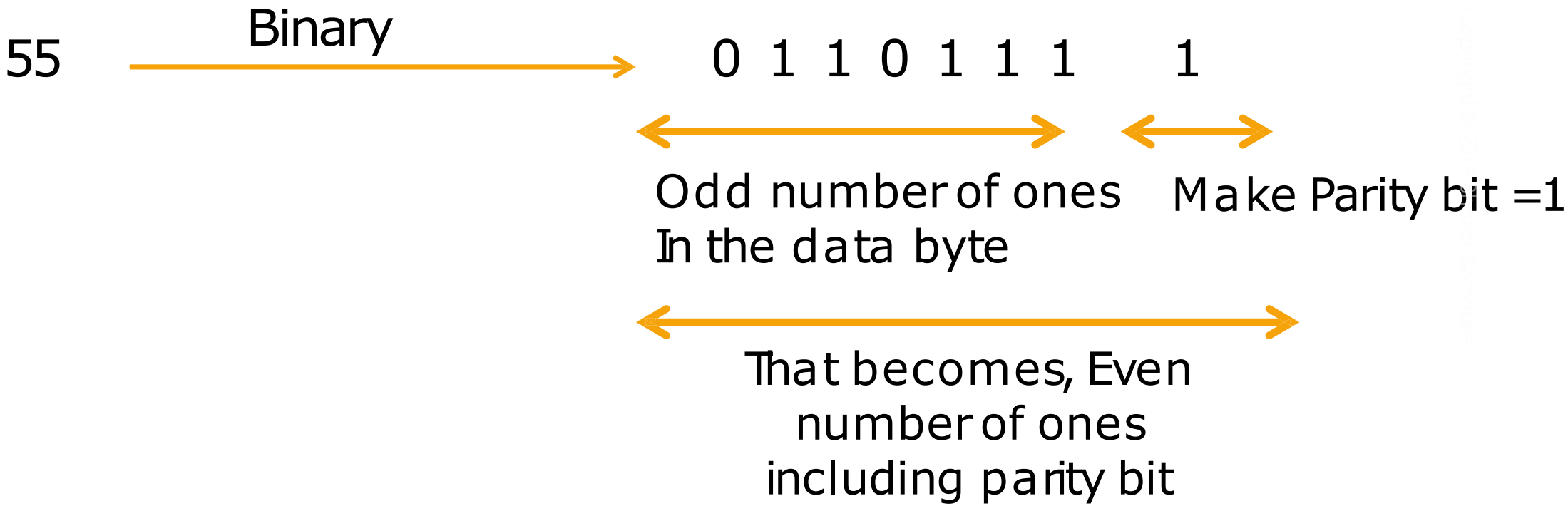
- This value determines how long the transmitter holds a serial line high or low .


104 us
104 us
104 us
104 us
Start
bit 0
bit 1
bit 2

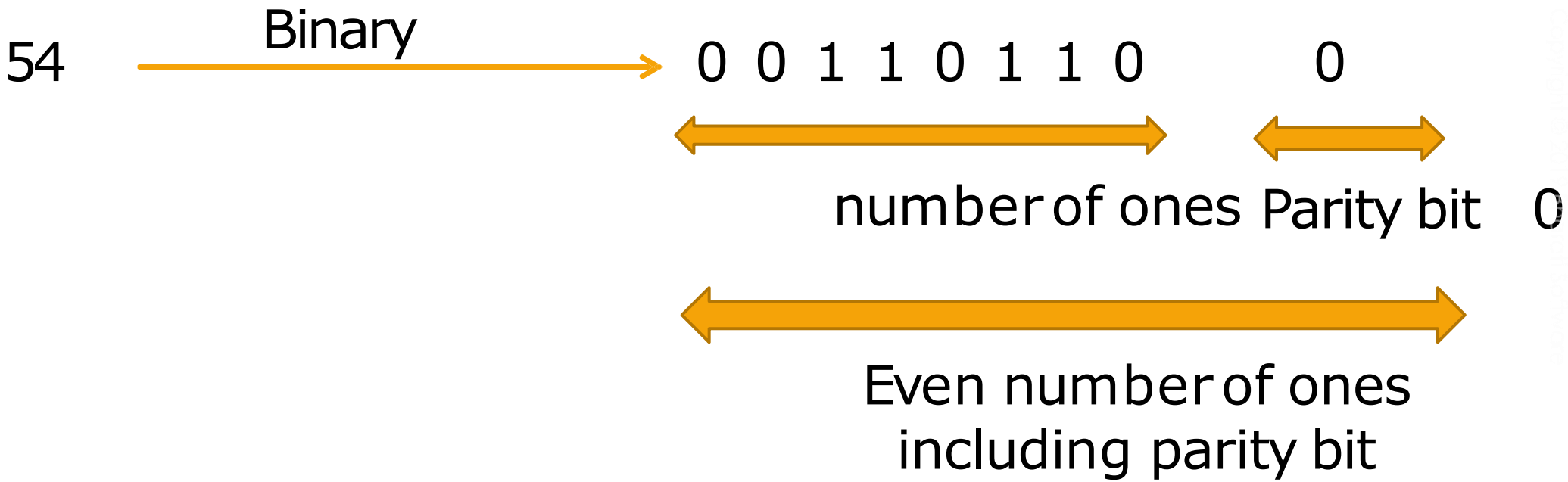
Adding Parity bit is a simplest method of error detection. Parity is simply the number of ones appearing in the binary form of a number.



Even parity

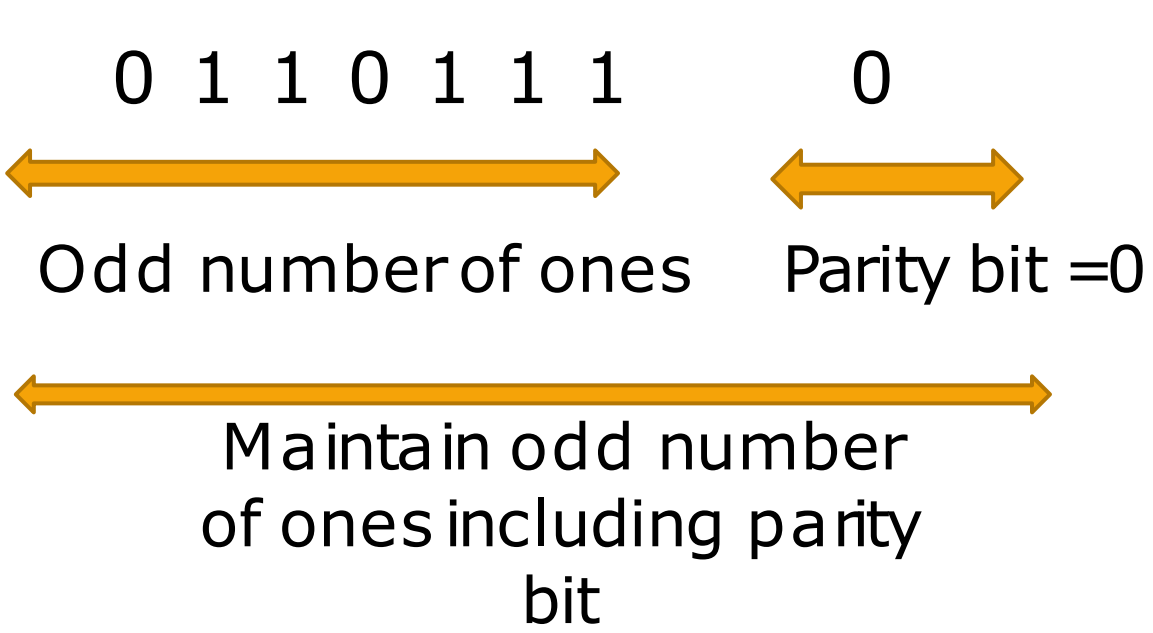

55
Binary
0 1 1 0 1 1 1
1
Odd number of ones
In the data byte
Make Parity bit =1
That becomes, Even
number of ones
including parity bit

Even parity

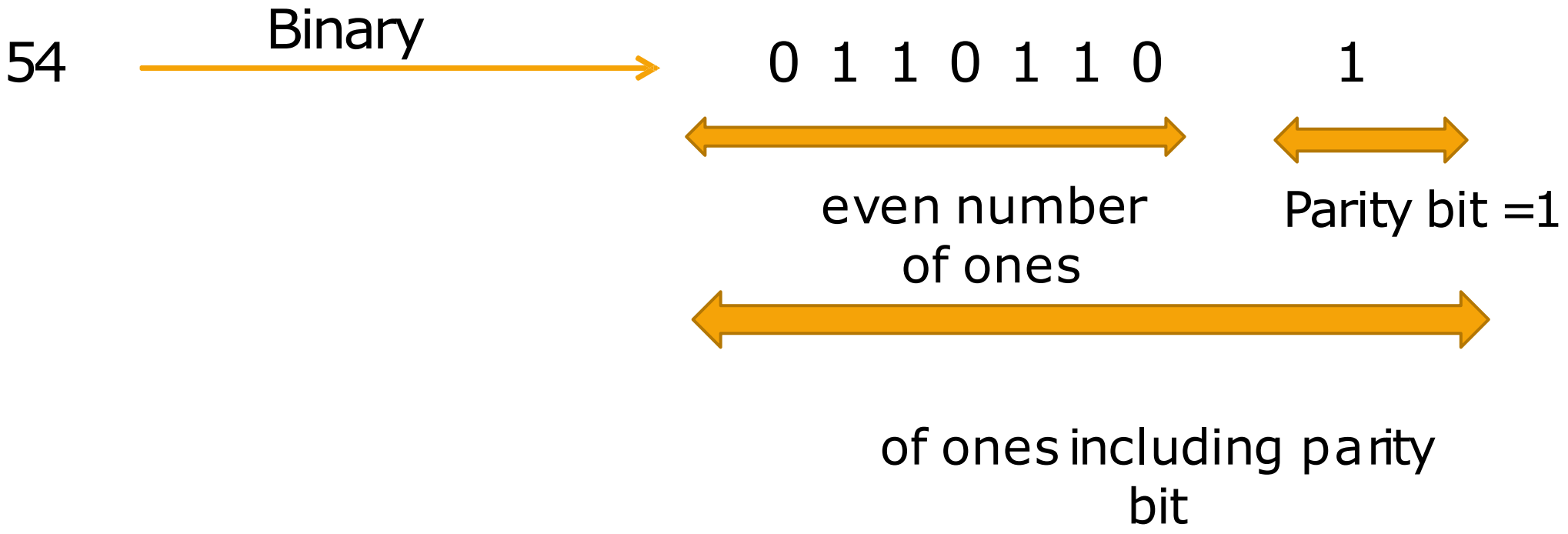

54
Binary
0 0 1 1 0 1 1 0
0
number of ones
Parity bit
0
Even number of ones
including parity bit

Odd parity

55 Binary →



Odd parity


54
Binary
0 1 1 0 1 1 0
1
even number of ones
Parity bit =1
of ones including parity bit

- Even parity results in even number of 1s, whereas odd parity results in odd number of 1s, when counted including the parity bit.

Baud Rate

$$Baud\ Rate = \frac{f_{PCLK}}{8 \times (2 - OVER8) \times USARTDIV}$$

- If *OVER8* is 0, then the signal is oversampled by 16, and 4 bits are used for the fractional part.
- If *OVER8* is 1, then the signal is oversampled by 8, and 3 bits are used.
- If BRR is **0x1BC** and *OVER8* is 0, then **0x1B** is the integer part and **0xC** is the fractional part.
- $USARTDV = 0x1B + \frac{0xC}{0x10} = 27 + \frac{12}{16} = 27.75$

Baud Rate

- Suppose the processor clock f_{PCLK} is 16MHz, and the system is oversampled by 16 ($OVER8 = 0$),
- $USARTDIV = \frac{f_{PCLK}}{8\times(2-OVER8)\times Baud\ Rate}$

$$= \frac{16 \times 10^6}{8 \times (2 - 0) \times 9600} = 104.1667$$

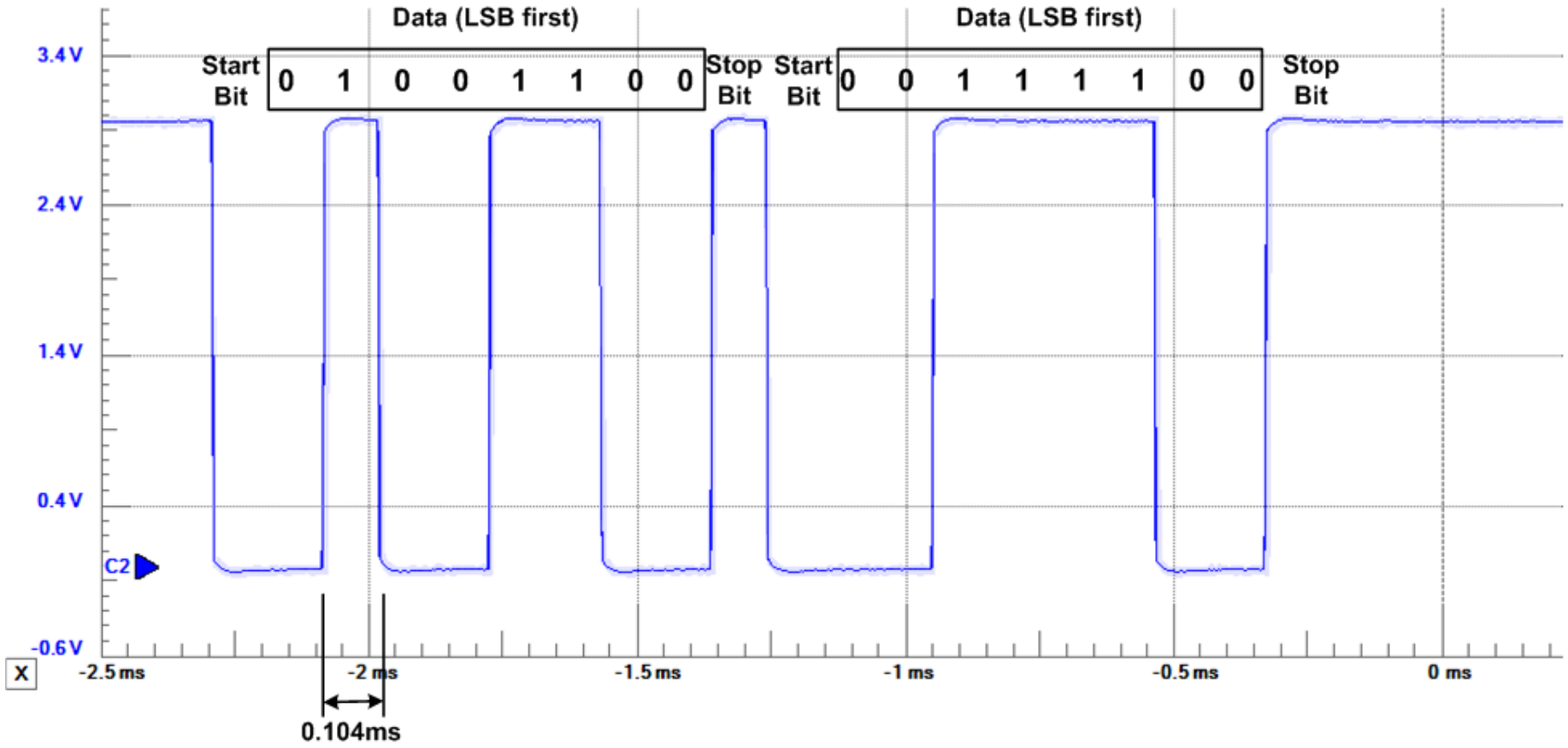
- Thus *USARTDIV* is 104.1875, which is encoded as 0x683.
- desired baud rate 9600

$$Baud\ Rate = \frac{16 \times 10^6}{8 \times (2 - 0) \times 104.1875} = 9598$$

Error Detection

- Even Parity: total number of "1" bits in data and parity is even
- Odd Parity: total number of "1" bits in data and parity is odd
- Example: Data = 10101011 (five "1" bits)
  - The parity bit should be 0 for odd parity and 1 for even parity
- This can detect single-bit data corruption

Transmitting 0x32 and 0x3C

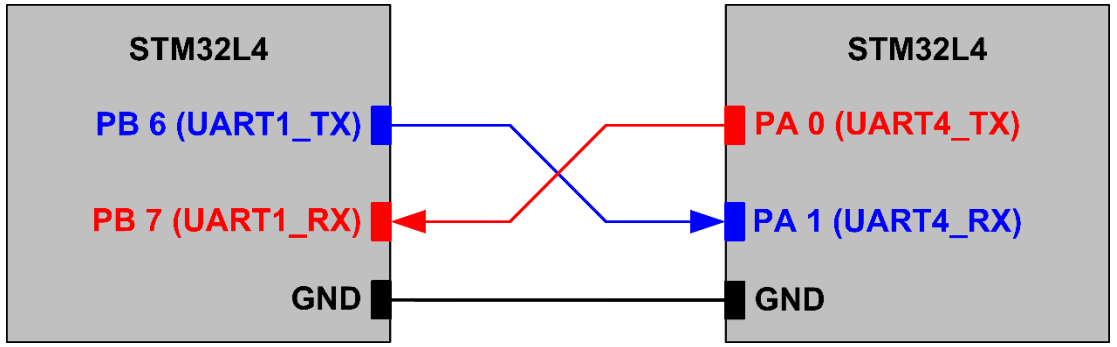


$$\frac{1}{9600} = 0.10417 \times 10^{-3}$$

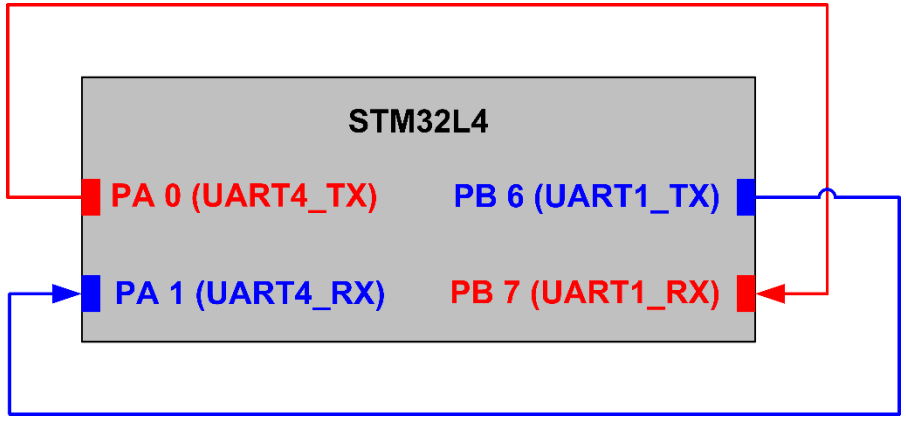
1 start bit, 1 stop bit, 8 data bits, LSB first, no parity, baud rate = 9600

UART Connection

Two boards:



Single board:



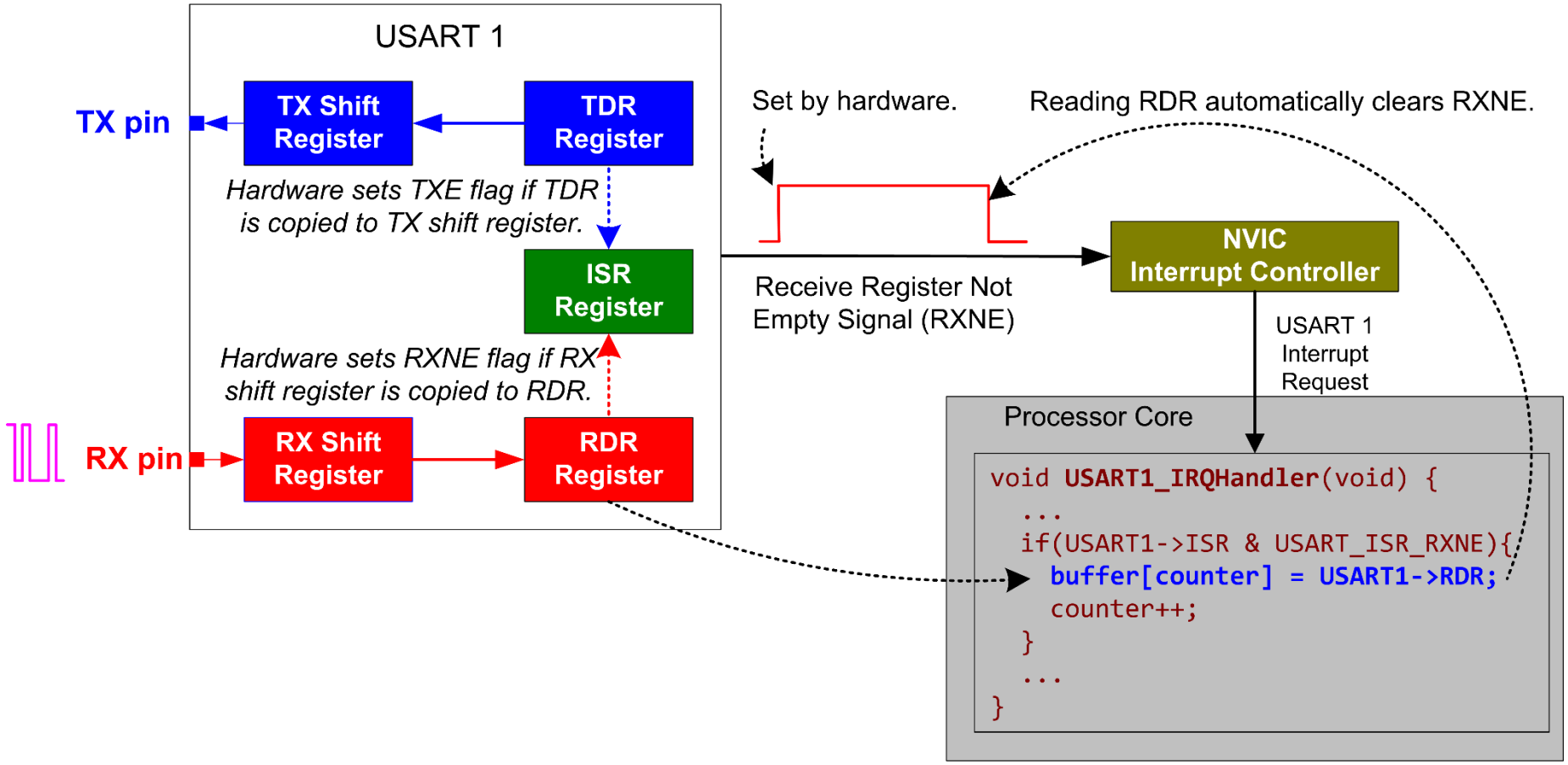
Sending Data

```
void USART_Write(USART_TypeDef * USARTx, uint8_t * buffer, int nBytes) {
   int i;

   // TXE is cleared by a write to the USART_DR register.
   // TXE is set by hardware when the content of the TDR register has been transferred into the
   // shift register.

   for (i = 0; i < nBytes; i++) {
    // wait until TXE (TX empty) is set
    // Writing USART_DR automatically clears the TXE flag
      while (!(USARTx->ISR & USART_ISR_TXE));
      USARTx->DR = (buffer[i] & 0x1FF);
   }

   while (!(USARTx->ISR & USART_ISR_TC));     // Wait until TC bit is set
   USARTx->ICR &= ~USART_ICR_TCCF;            // Write 1 to clear the TC flag
}
```

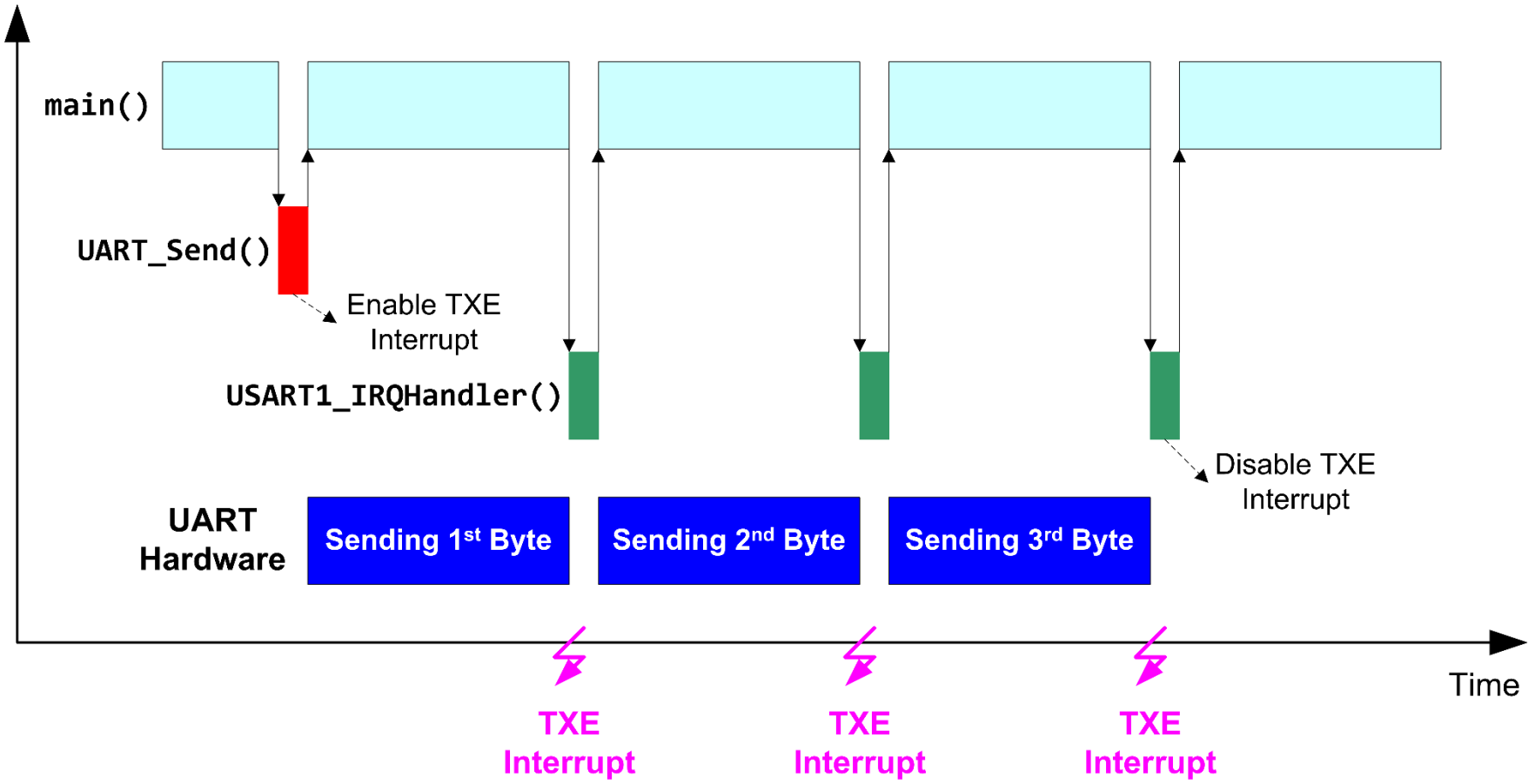
Receiving Data

```
void USART_IRQHandler(USART_TypeDef * USARTx, uint8_t * buffer,  uint8_t * pRx_counter){

   if(USARTx->ISR & USART_ISR_RXNE) { // Received data

     buffer[*pRx_counter] = USARTx->DR; // Reading USART_DR automatically clears the RXNE flag

     (*pRx_counter)++;

     if((*pRx_counter) >= BufferSize )
       (*pRx_counter) = 0;
   }
}

void USART1_IRQHandler(void) {
   USART_IRQHandler(USART1, USART1_Buffer_Rx, &Rx1_Counter);
}

void USART2_IRQHandler(void) {
   USART_IRQHandler(USART2, USART2_Buffer_Rx, &Rx2_Counter);
}
```

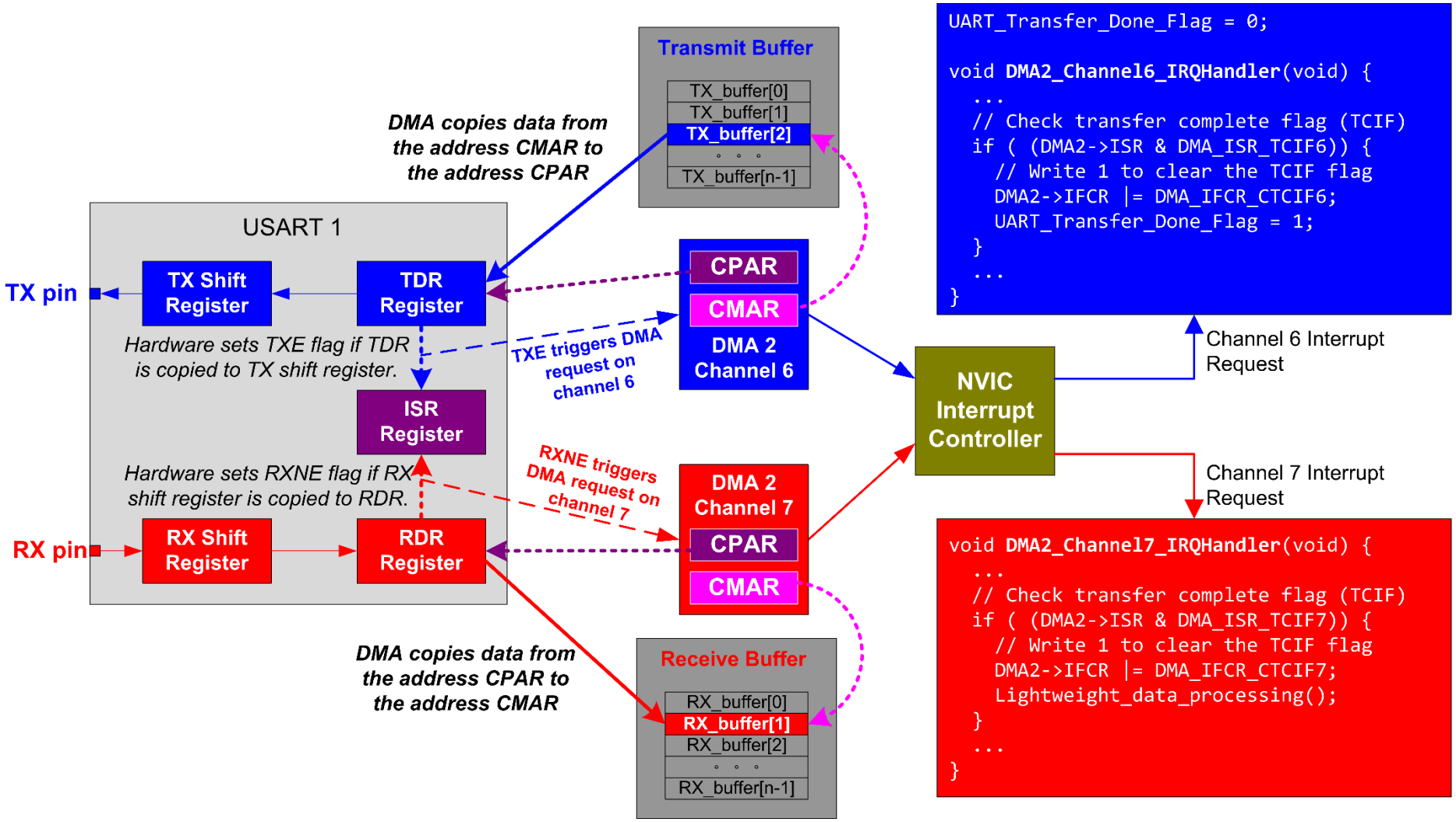
UART Interrupt: Receiving Data



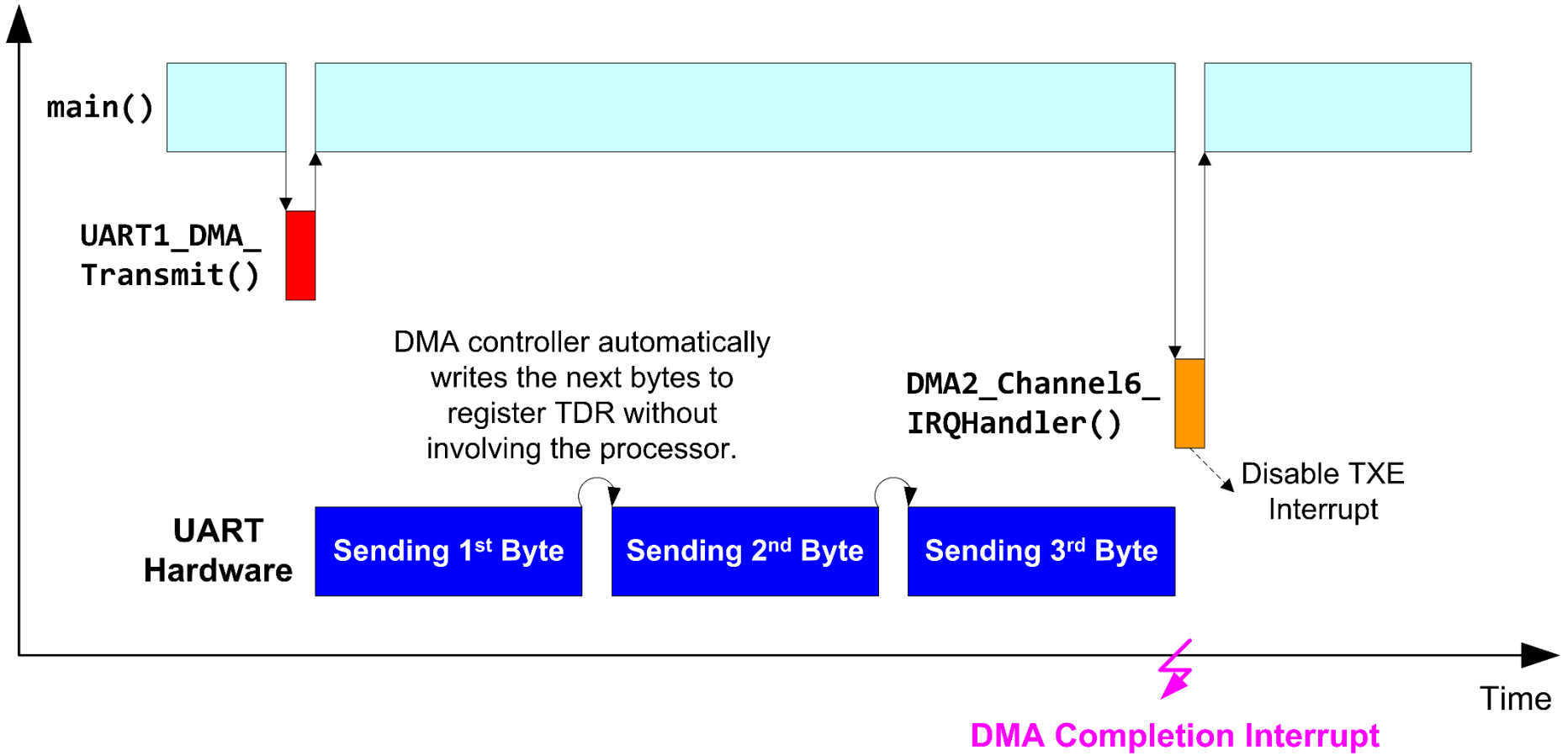
UART Interrupt: Receiving Data



UART DMA: Receiving & Sending



UART DMA: Receiving & Sending



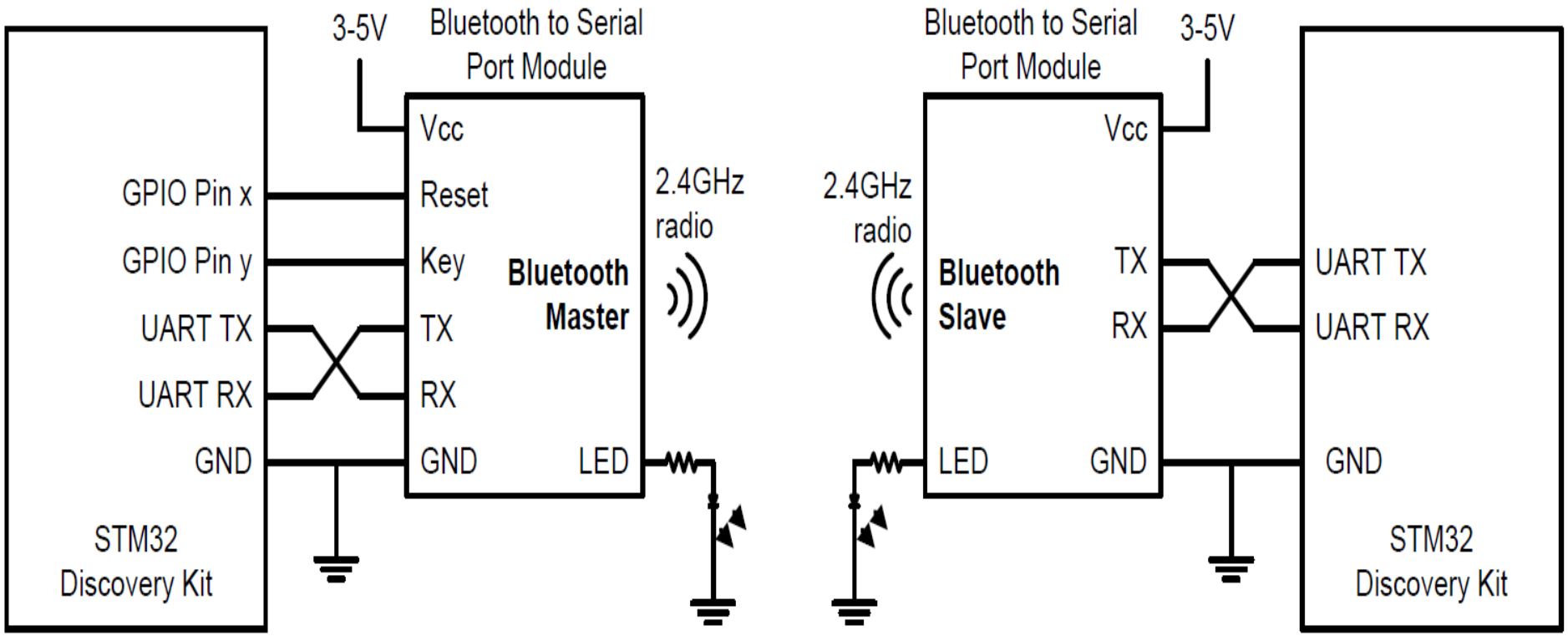
Voltage Levels

Standard	Voltage signal	Max distance	Max speed	Number of devices supported per port
RS-232	Single end (logic 1: +5 to +15V, logic 0: -5 to -15 V)	100 feet	115Kbit/s	1 master, 1 receiver
RS-422	Differential (-6V to +6V)	4000 feet	10Mbit/s	1 master, 10 receivers
RS-485	Differential (-7V to +12V)	4000 feet	10Mbit/s	32 masters, 32 receivers

Standards:

- How far can signal transfer?
- How fast can it transfer data?
- How many devices supported per port?
- How many masters allowed?

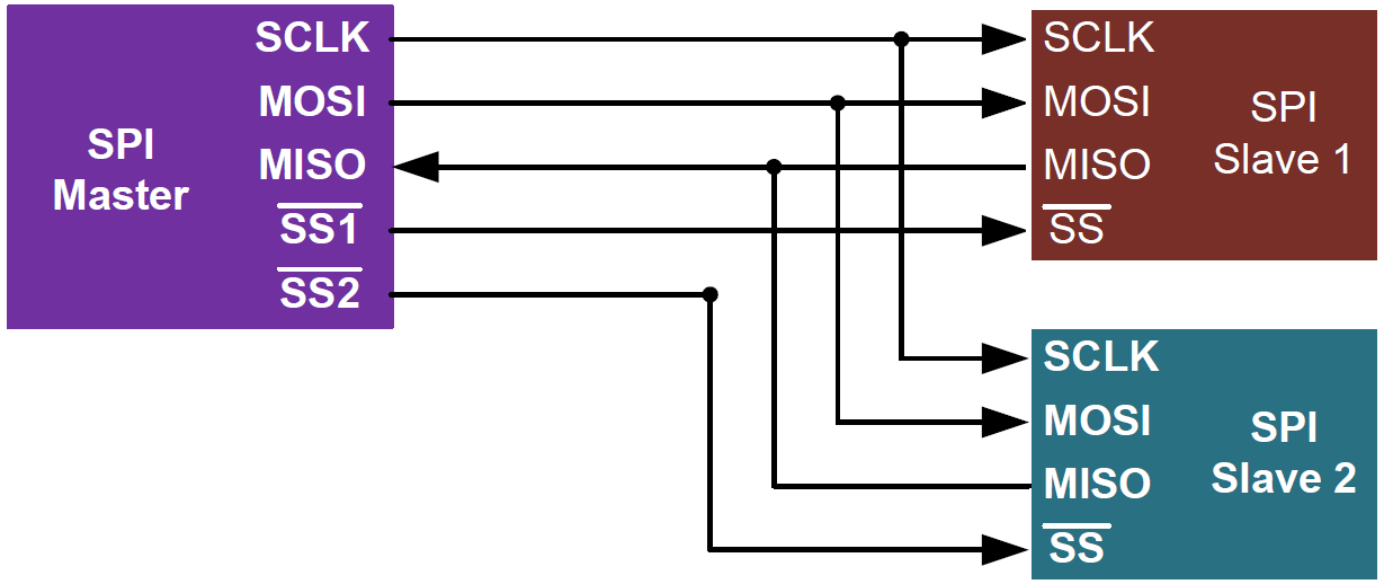
Bluetooth


3-5V
Bluetooth to Serial
Port Module
Vcc
Reset
Key
TX
RX
GND
LED
Bluetooth
Master
2.4GHz
radio
GPIO Pin x
GPIO Pin y
UART TX
UART RX
GND
STM32
Discovery Kit
Bluetooth to Serial
Port Module
3-5V
Vcc
2.4GHz
radio
Bluetooth
Slave
TX
RX
LED
GND
UART TX
UART RX
GND
STM32
Discovery Kit

SPI: Serial Peripheral Interface

Serial Peripheral Interface (SPI)

- Synchronous full-duplex communication
- Single master, multiple slaves
- No hardware flow control or slave acknowledgment
- Slave cannot communicate with slave directly
- Higher throughput than I2C



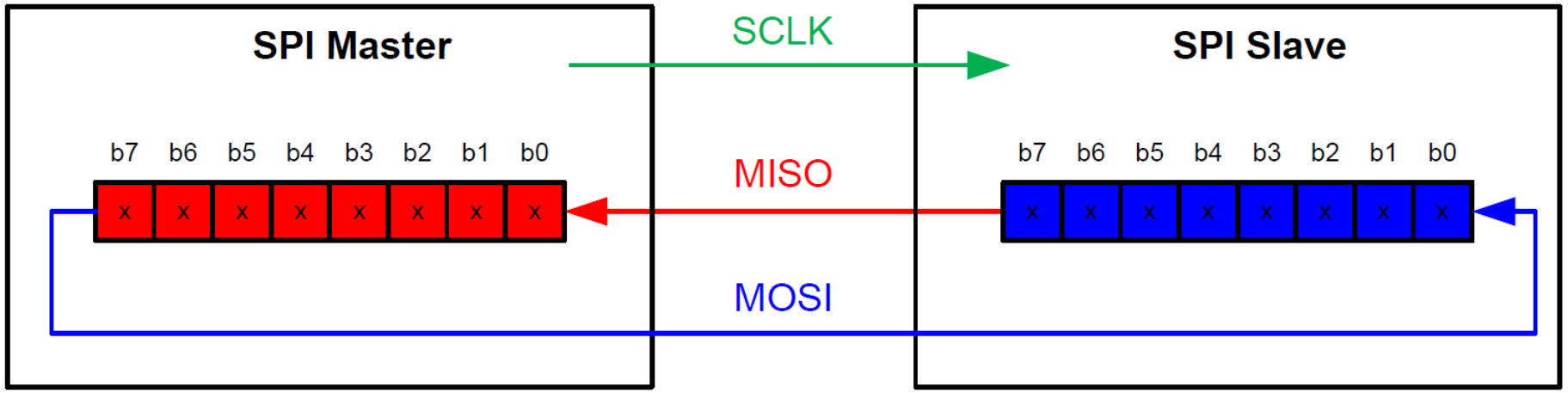
SCLK: serial clock

MOSI: master out slave in

SS: slave select (active low)

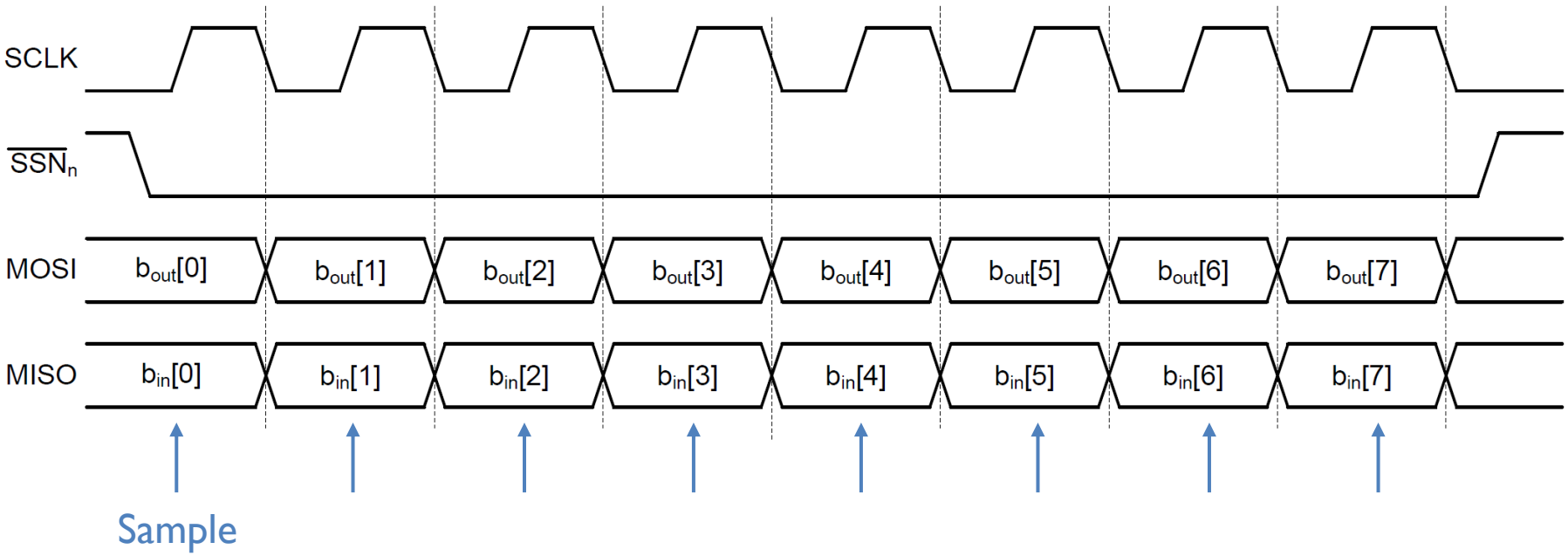
MISO: master in slave out

SPI Synchronous Data Exchange

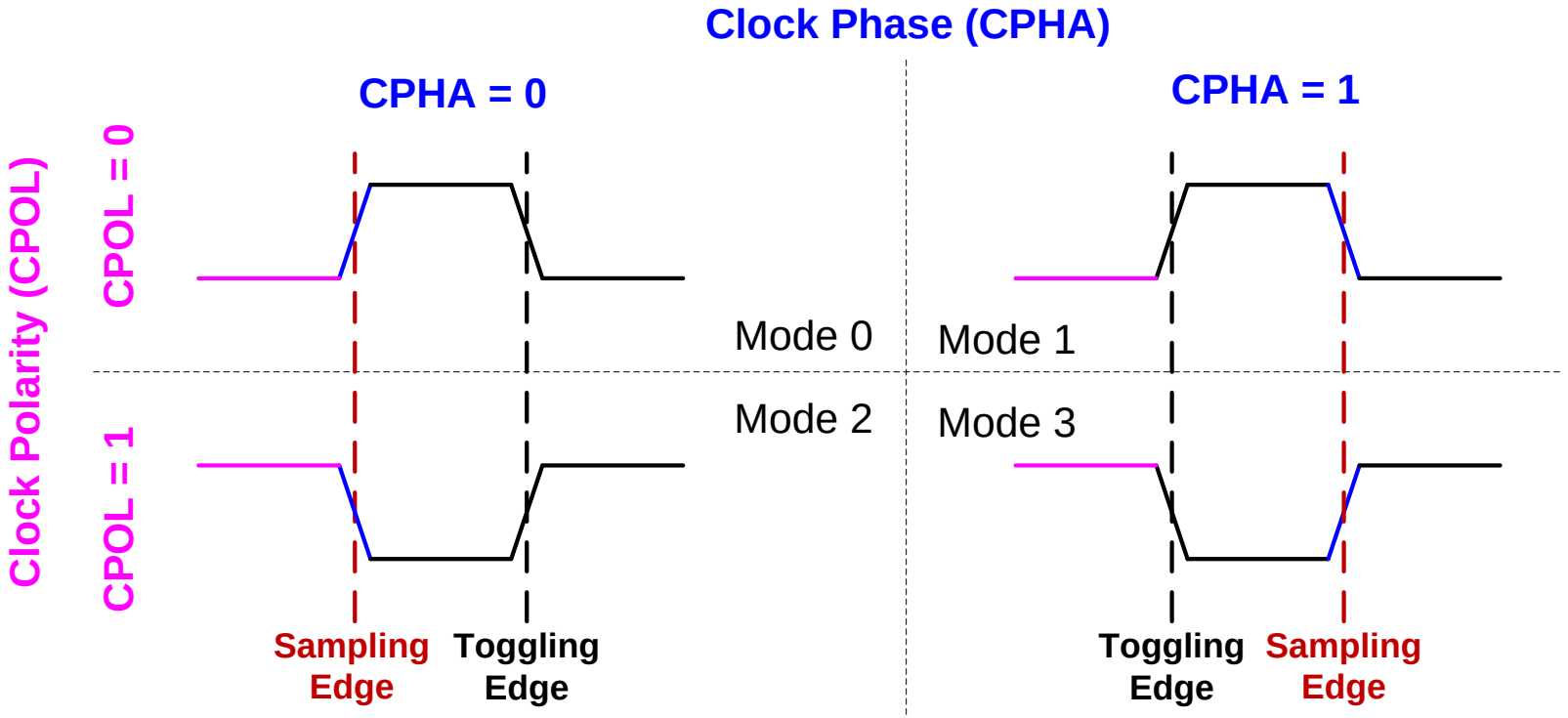


- Master has to provide clock to slave
- *Synchronous exchange*
  - Master shift out a bit to slave, and shifts in a bit from slave.
- Only master can start the data transfer. Thus no arbitration required.

SPI Timing Diagram

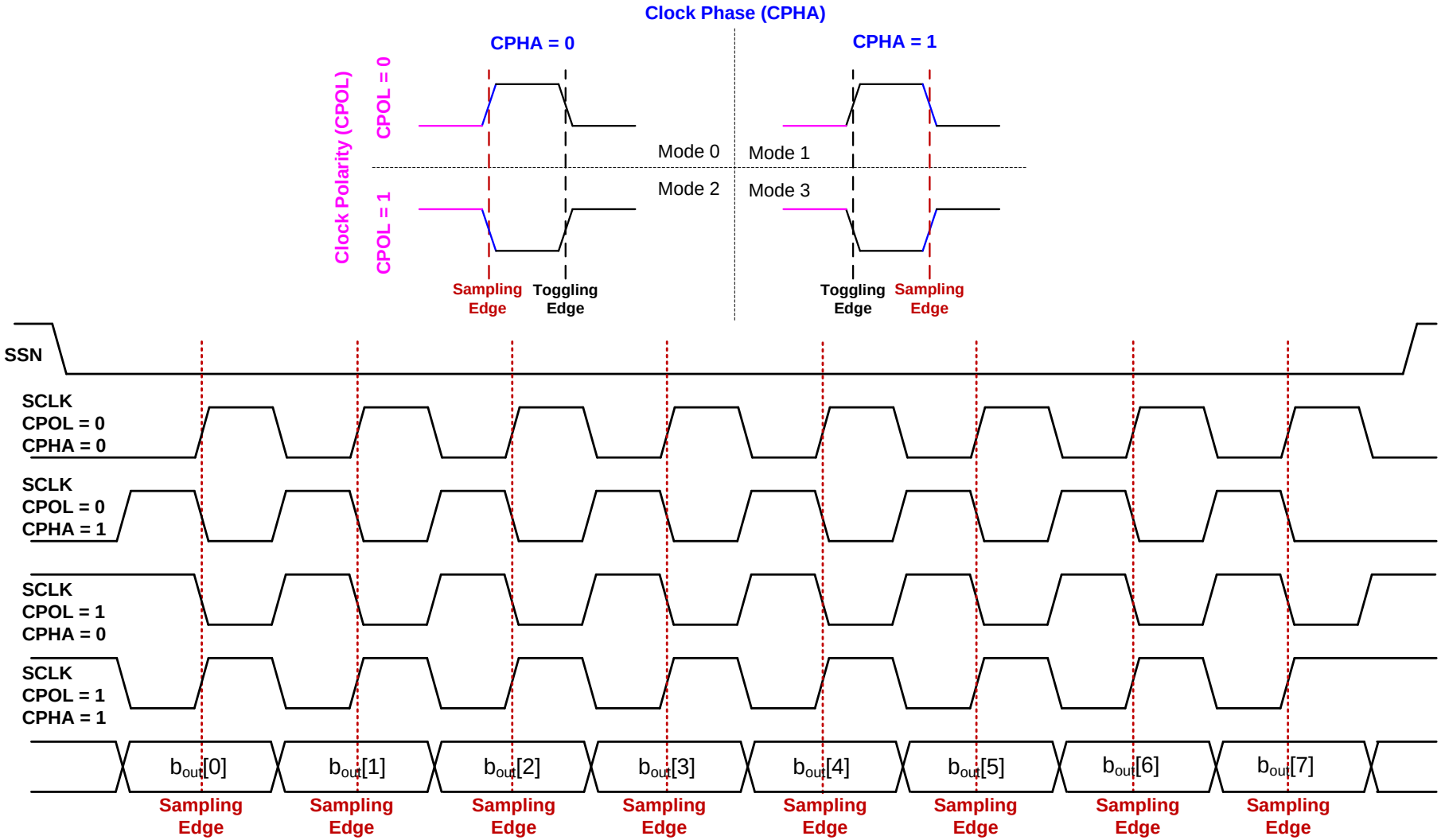


SPI Clock Phase and Polarity

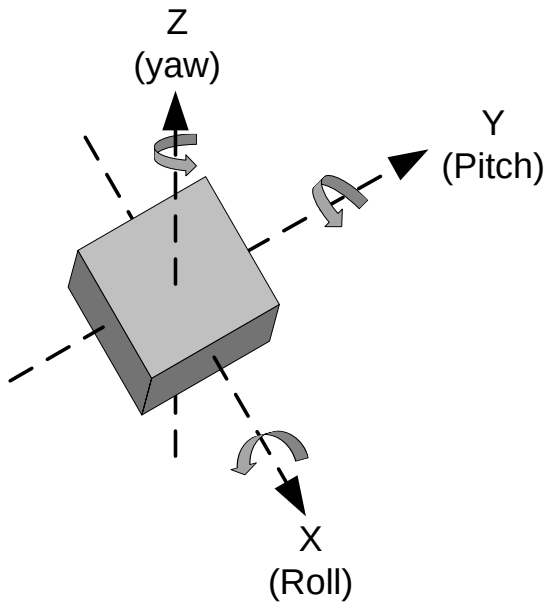


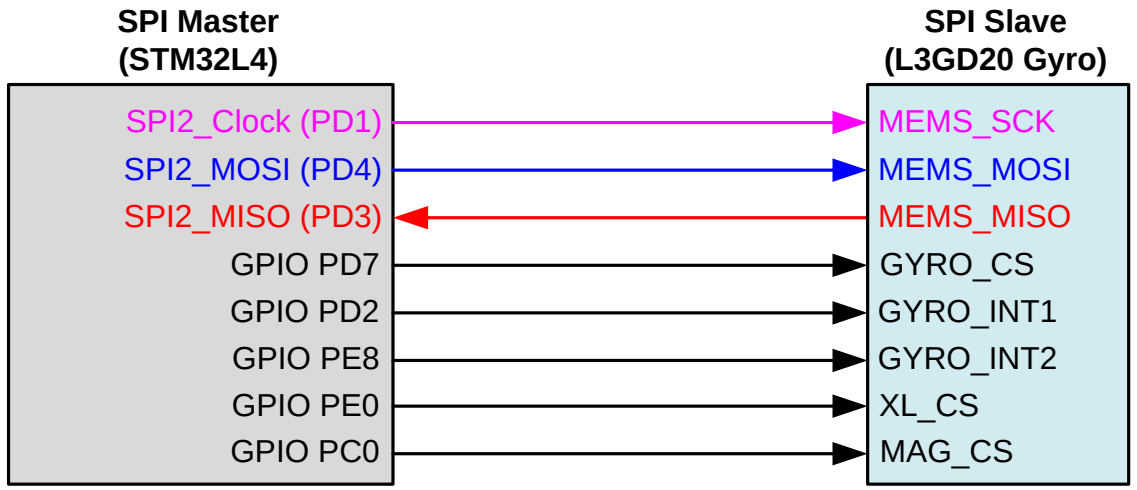
- Combination of CPOL and CPHA determines the clock edge for transmitting and receiving.
- CPOL = 0 ⟶ SCLK is pushed to low during idle. Otherwise, pulled to high during idle.
- CPHA = 0 ⟶ the first clock transition (either rising or falling) is the first data capture edge. Otherwise, the second clock transition is the first data capture edge.

SPI Clock Phase and Polarity

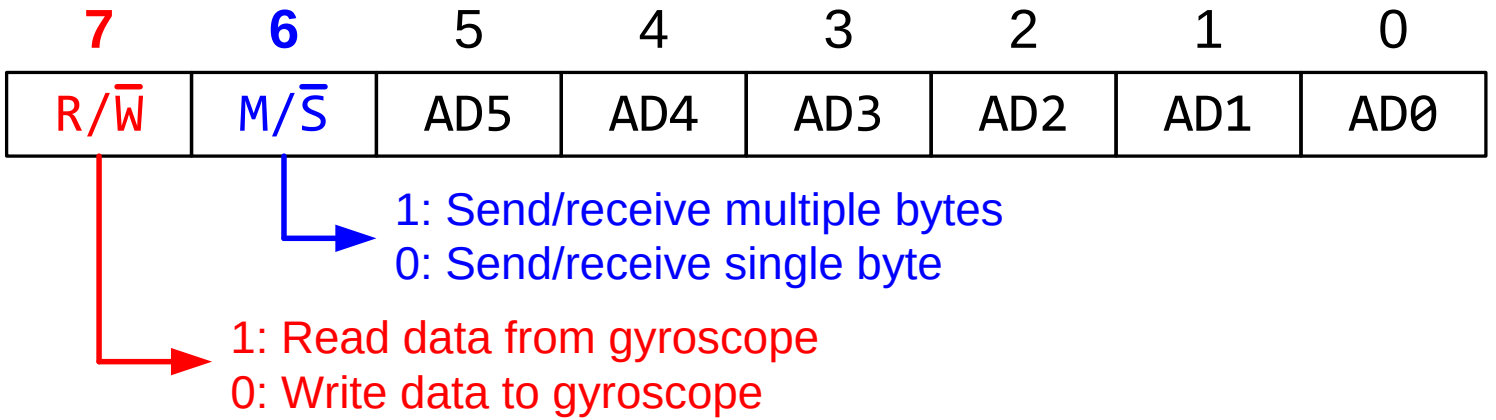


Gyro Sensors





L3GD20


7
6
5
4
3
2
1
0
R/W
M/S
AD5
AD4
AD3
AD2
AD1
AD0
1: Send/receive multiple bytes
0: Send/receive single byte
1: Read data from gyroscope
0: Write data to gyroscope

SPI vs I2C

- SPI by Motorola and I2C by Philips
- Both are synchronous protocols for short distance communications
- Generally, operates on 3.3 V or 5V

	SPI	I2C
Advantages	• Faster speed. Can be 10 Mbps or more. • Full duplex	• Simplicity. Commonly used as a 2-wire bus. • Adding new slave is easy