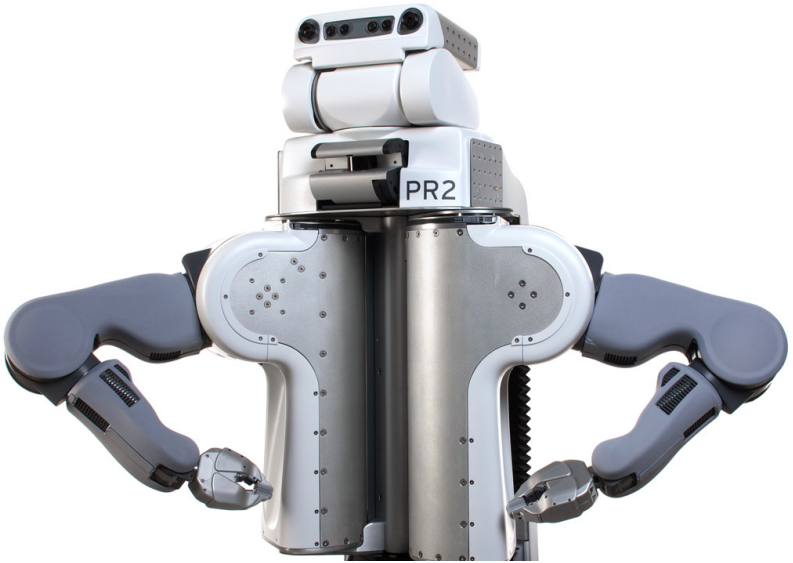
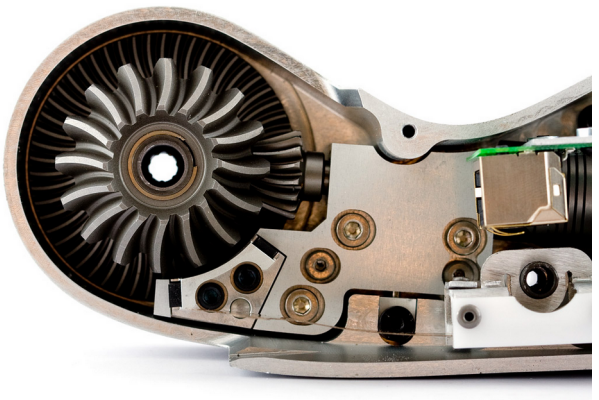
Robot Operating System
Introduction to Hardware Interfacing

Agenda

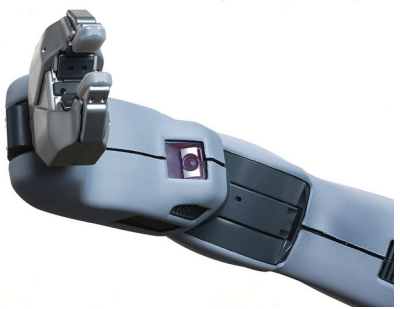
- Types & Need for Hardware
- Communication requirement
- Overview Serial Communication
- Serial Communication Hardware Demo 1 - Arduino
- Open Interface Protocol Hardware Demo 2 - Arduino
- Open Interface Protocol Hardware Demo 3 – ROS + Arduino

Robot Component Classification





https://youtu.be/c3Cq0sy4TBs



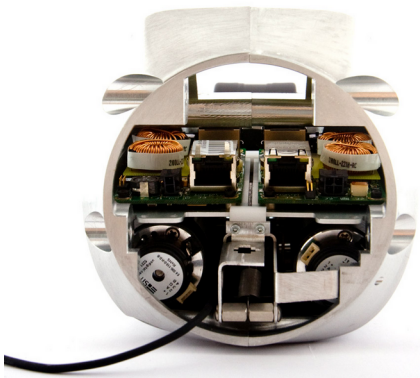


Image courtesy: https://robots.ieee.org/robots/pr2

Classification of Robot Components / Hardwares

Sensing	: used to perceive the world,
Actuation	: allow interaction with the environment,
Communication	: provide a means of interconnection,
Cognition	: The brain of the robot
Components	: that group together different sub-components

Example Hardware Hilights

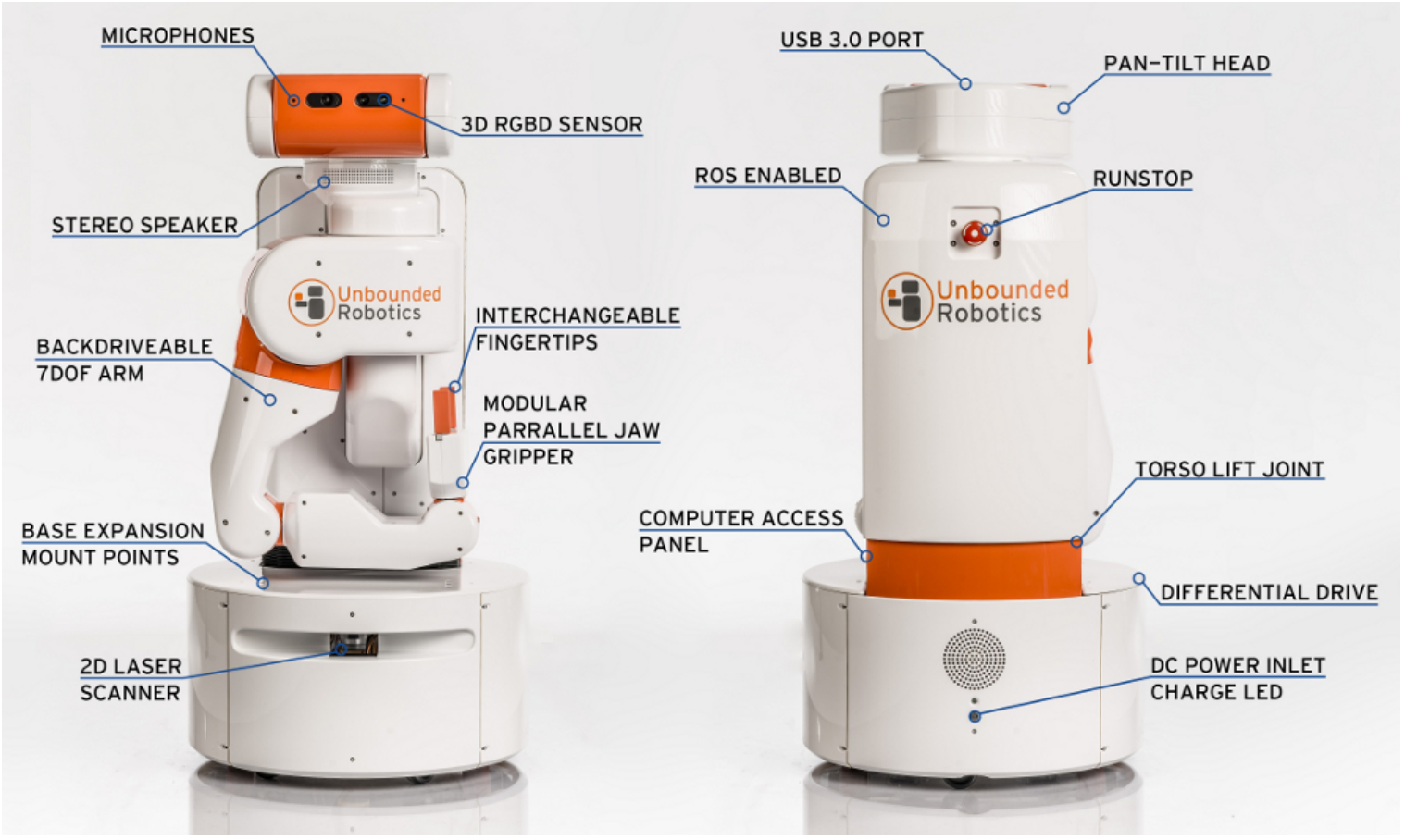


Image courtesy: https://robohub.org/its-official-you-can-now-buy-an-ubr-1/

ROS Control

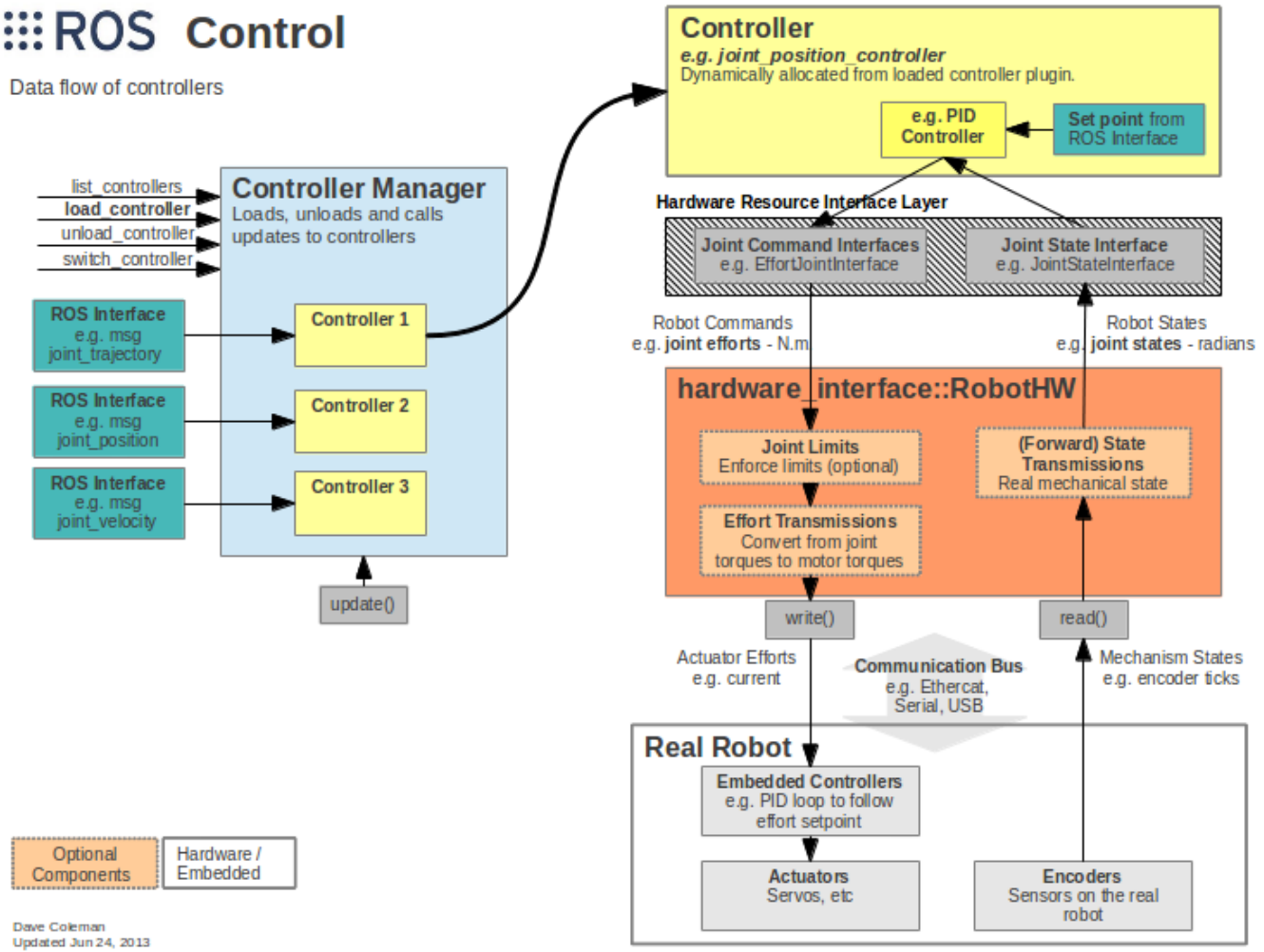


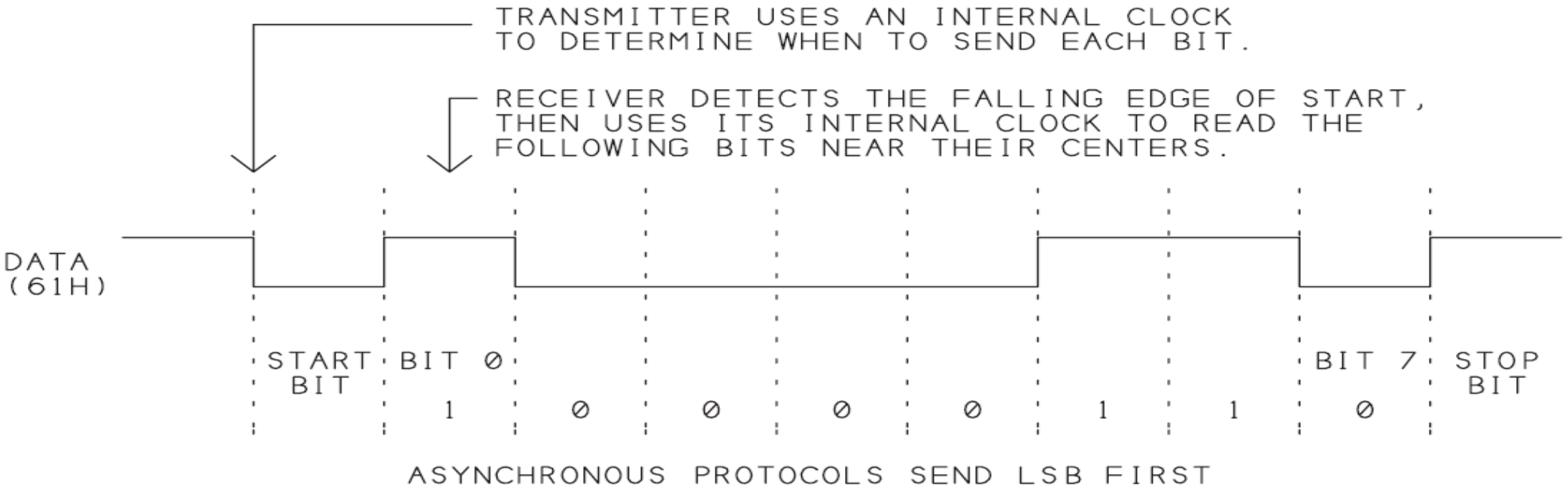
Image courtesy: http://wiki.ros.org/ros_control

Communications Interfaces for Robot Hardwares

- USB 2.0 /3.0 : Camera, IMU, Laser Sensor, Depth Camera & many more.
- Serial : Inertial Mmeasurement Unit, Actuators and ToF Sensors
- Ethernet : Actuators, Camera, Laser Sensor, Depth Camera & more
- Ethercat : Actuators & camera

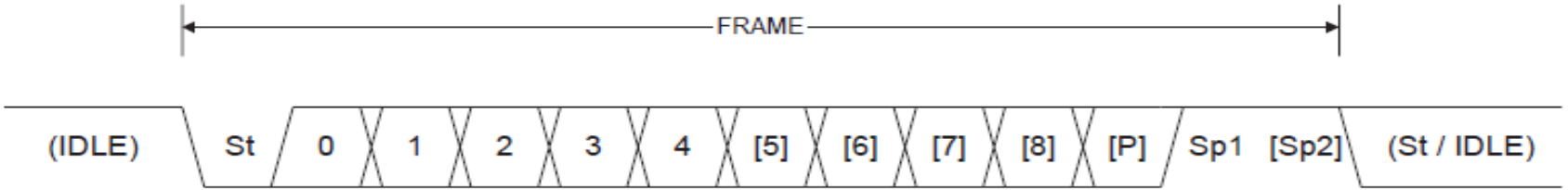
EIA-232 Interface Standard

- UART transmits data in chunks often called ***words***
- Each word contains a ***Start bit, data bits, an optional parity bit, and one or more Stop bits***
- ***LSB*** data bits is transmitted ***first***



EIA-232 Interface Standard (TTL Logic)

- UART support multiple word formats
- ***Start bit*** is always low and ***Stop bit*** (1, 1.5, 2) always high



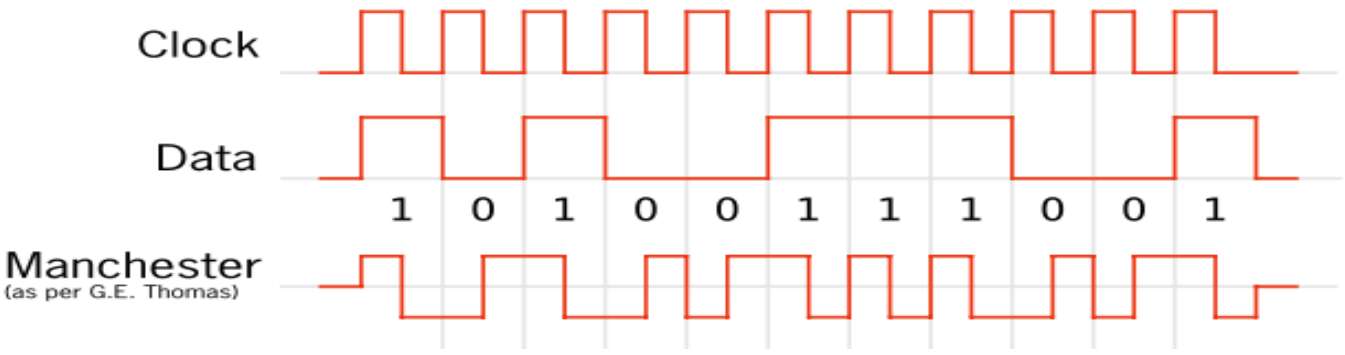
- ***Data bits*** vary from 5 to 8 bits

- ***Parity bit*** can provide a ***basic form of error detecting***, and the formats that use the parity bit can use a parity type of even, odd, mark, or space
- Parity bit is given by ***exclusive-or of all the data bit***

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

EIA-232 Interface Standard

- Gross bit rate/ raw bit rate/data signaling rate/gross data transfer rate
  R_b = 1/Bit transmission time (T_B) *(Bits per Second)*
- *N* bits conveyed per symbol, N = 1 for NRZ
- Baud rate/ Symbol rate (f_s)= R_b/N



RZ or RTZ

Baud rate	Cable length (meters)
110	850
300	800
600	700
1200	500
2400	200
4800	100
9600	70
19200	50
115 K	20

Arduino Simple Serial Program 1

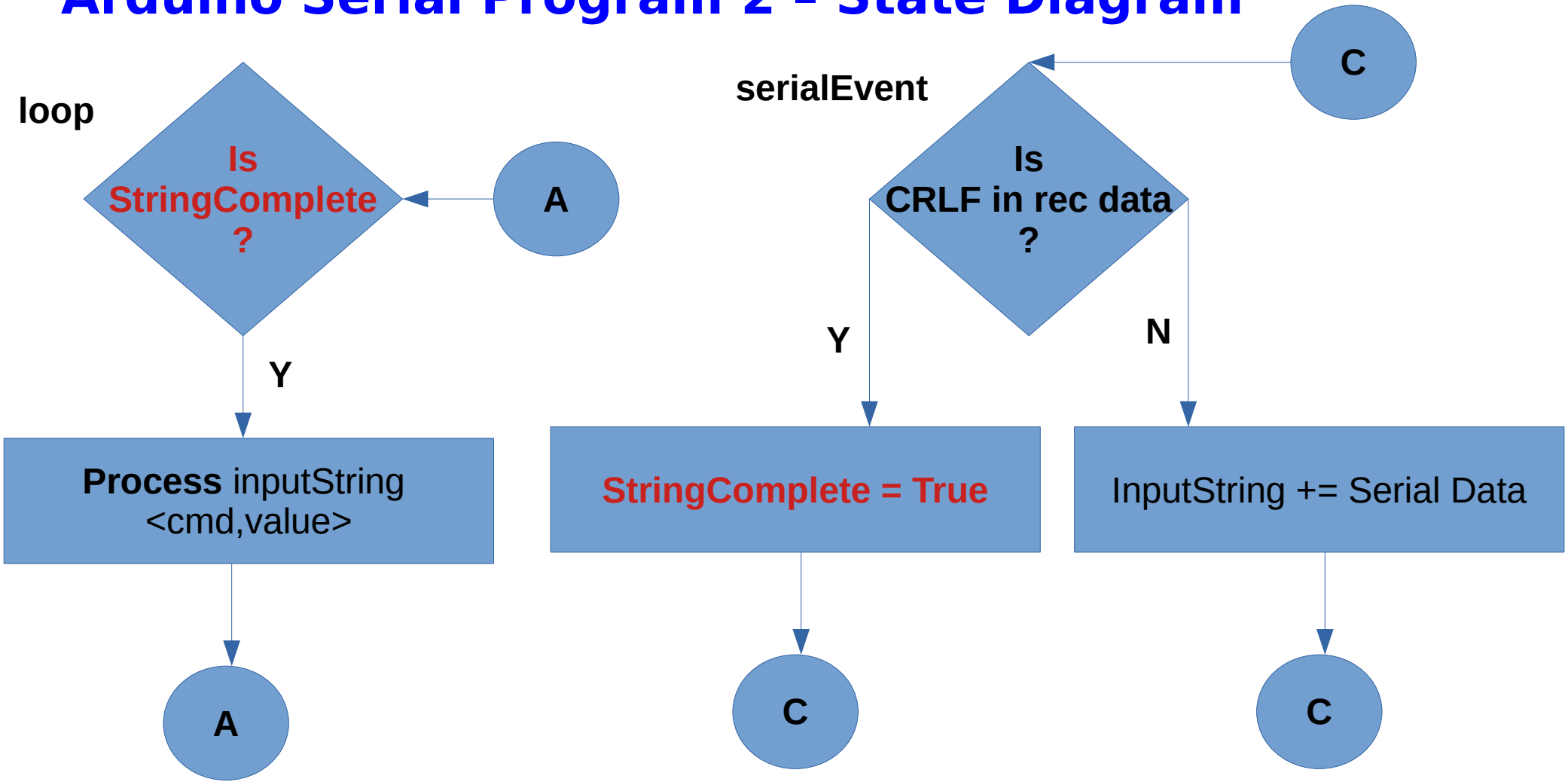
```
void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}


void loop() {
  // put your main code here, to run repeatedly:
  Serial.write("1");
  Serial.write(65);
  Serial.write("\r\n");
  delay(500);
}
```

Arduino Simple Serial Program 2

<command:value>

Arduino Serial Program 2 – State Diagram



Arduino Serial Program 2 ...

```
String inputString = "";
bool stringComplete = false;
String command = "";

void setup() {
  // initialize serial:
  Serial.begin(9600);
  inputString.reserve(200);
}
```

```
void loop() {
  if (stringComplete) {
    Serial.println(inputString);

    int stloc = inputString.indexOf('<');
    int endloc = inputString.indexOf('>',stloc + 1);
    if(stloc >=0 and endloc >stloc)    {
      String data = inputString.substring(stloc + 1,endloc);
      Serial.println(data);
      endloc = data.indexOf(':');
      command = data.substring(0,endloc);
      Serial.print("Command:");
      Serial.println(command);
    }
   else    { Serial.println("Command:Error");     }
    inputString = "";
    stringComplete = false;
  }
}
```

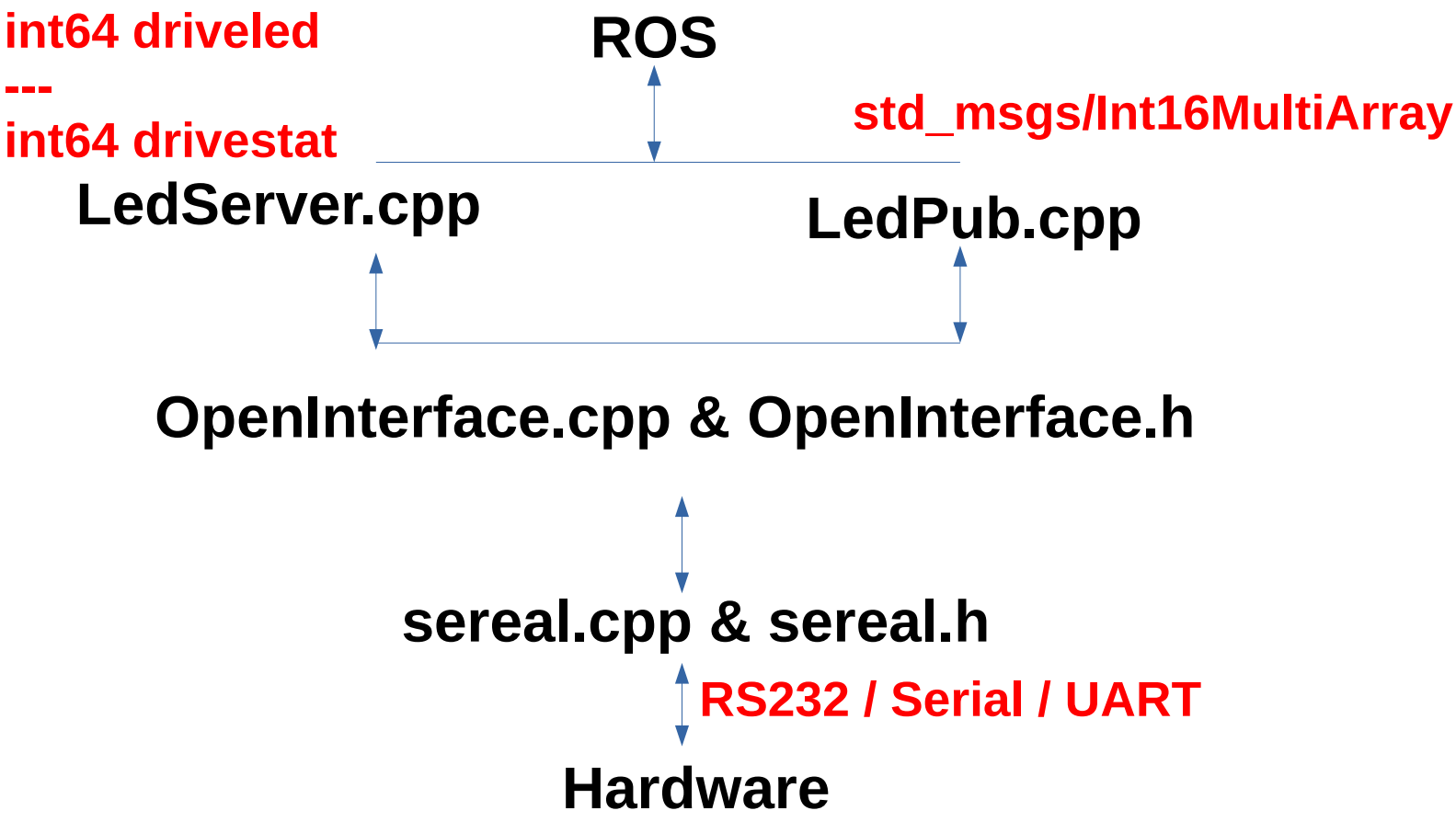
Arduino Serial Program 2 ...

```
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag so the main loop can
    // do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

Arduino Program 3 – Open Interface Protocol

- Led On <l:1>
- Led Off <l:0>
- Stream Off <s:0>
- Stream On <s:1>
- Analog Data <a[ch]:[value]>
- Count Data <c:[count]>

Arduino Program 3 – Open Interface Protocol



Source Code & PPT can be downloaded from

https://bitbucket.org/vasanthbe/psg_fdp/src/master/

Thank you ...

Appendix 1:
Top 10 ROS based Industries in Market

Top 10 ROS-based robotics companies in 2019

1. Clearpath Robotics

- https://www.youtube.com/watch?v=PaSFFxj-9vI&feature=emb_logo

2. Fetch Robotics

- https://www.youtube.com/watch?v=CEIUrF7iOXk

3. Pal Robotics

- https://www.youtube.com/watch?v=xUeApfMAKAE&feature=emb_logo

4. Robotnik

- https://www.youtube.com/watch?v=zs0gH36RuEA&feature=emb_logo

5. Yujin Robots

- https://www.youtube.com/watch?v=t-KTHkbUwrU&feature=emb_logo

Top 10 ROS-based robotics companies in 2019

6. Robotis

- https://www.youtube.com/watch?v=B2pnXtooKOg&feature=emb_logo

7. Shadow Robot

- https://www.youtube.com/watch?v=V5W07j5vvlA&feature=emb_logo

8. Husarion

- https://www.youtube.com/watch?v=jlS3mzA4E2c&feature=emb_logo

9. Neobotix

- https://www.youtube.com/watch?v=qgqiD3oY5Iw&feature=emb_logo

10. Gaitech

- https://www.youtube.com/watch?v=yvDDQqJrYdw&feature=emb_logo

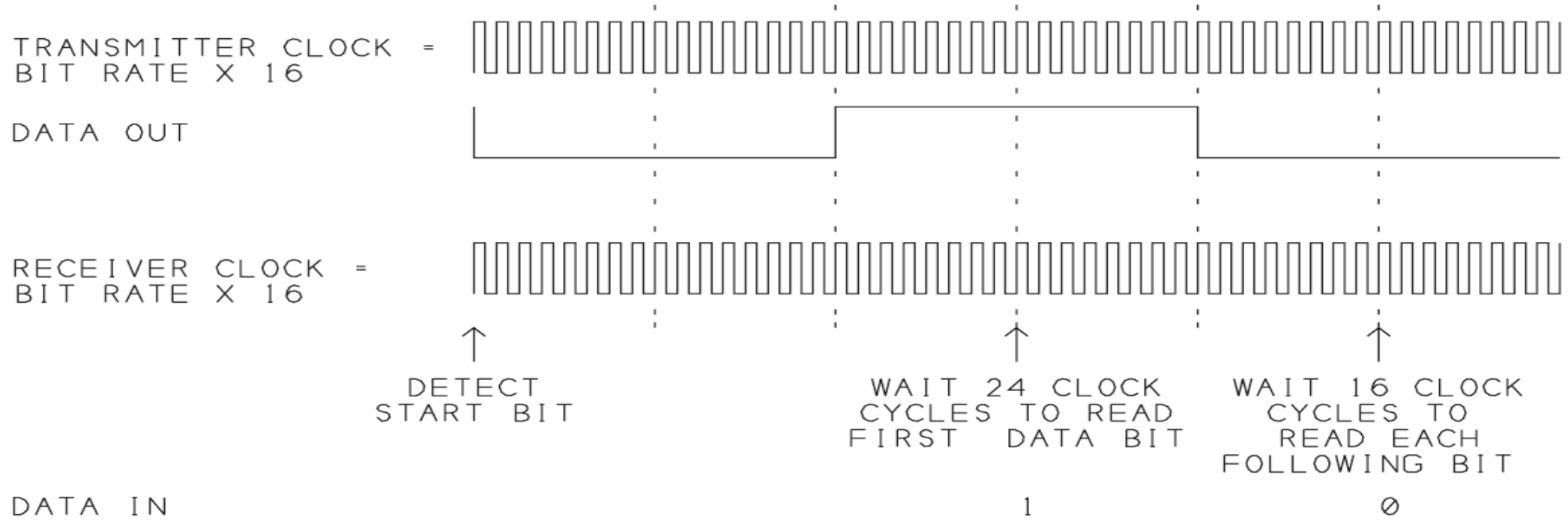
Appendix 2:
RS232 / EIA232 Information

EIA-232 Interface Standard

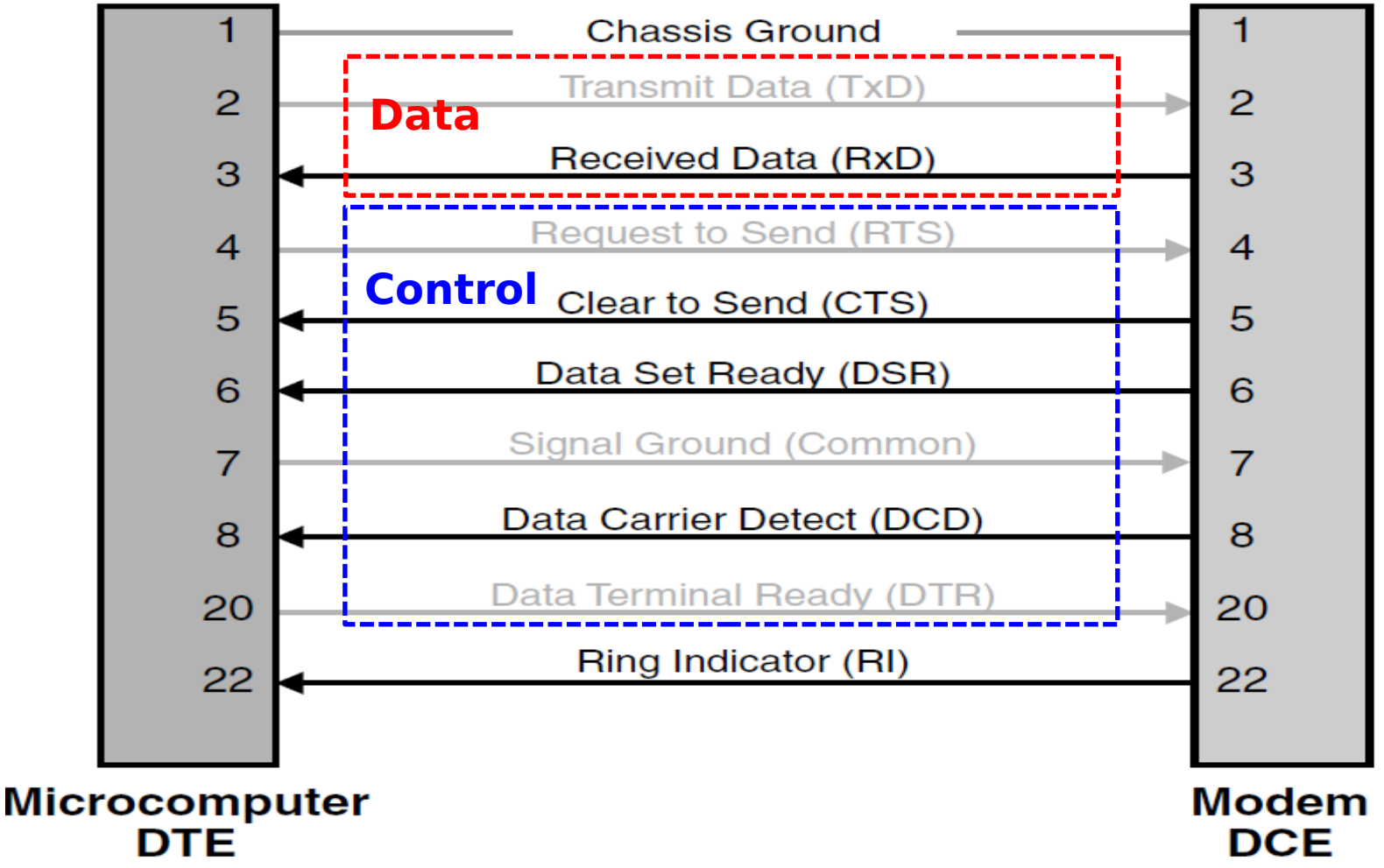
- Standard was developed for the single purpose of interfacing data terminal equipment (DTE) and data circuit terminating equipment (DCE) employing serial binary data interchange
- EIA-232 was originally named RS-232, (recommended standard)
- The prefix ‘RS’ was superseded by ‘EIA/TIA’ in 1988. The current revision is EIA/TIA-232E (1991),
- EIA standards define the electrical and mechanical details of the interface (layer 1 of the OSI model) and do not define a protocol
- Standard specifies the method of connection of two devices – the DTE and DCE
- ASCII encoding

EIA-232 Interface Standard

- receive clock with a frequency 16 times faster than the bit rate
- UART waits 16 clock cycles for the Start bit to end, then waits 8 more cycles to read bit zero in the middle of the bit.
- The UART reads bit that follows 16 clock cycles after previous bit



EIA-232 interface standard



EIA-232 : Electrical Signal Characteristics

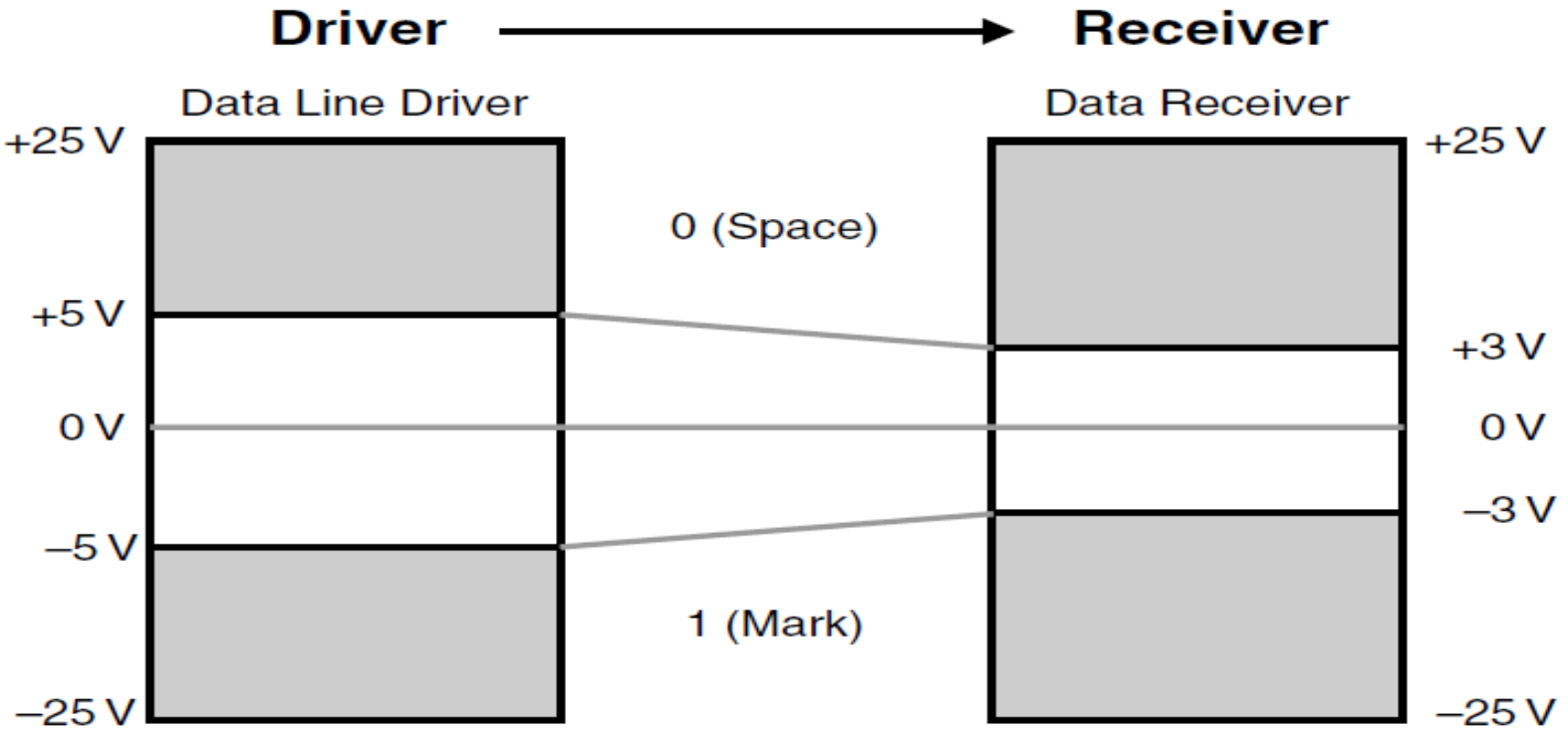
- ***The EIA-232 transmitter*** is required to produce voltages in the range +/− 15 to +/− 25 V as follows:
  - Logic 1: –5 V to –25 V
  - Logic 0: +5 V to +25 V
  - Undefined logic level: +5 V to –5 V

- At the ***EIA-232 receiver,*** the following voltage levels are defined:
  - Logic 1: –3 V to –25 V
  - Logic 0: +3 V to +25 V
  - Undefined logic level: –3 V to +3 V

- The rate at which voltage can ‘slew’ depends mainly on the cable capacitance and the capacitance increases with cable length

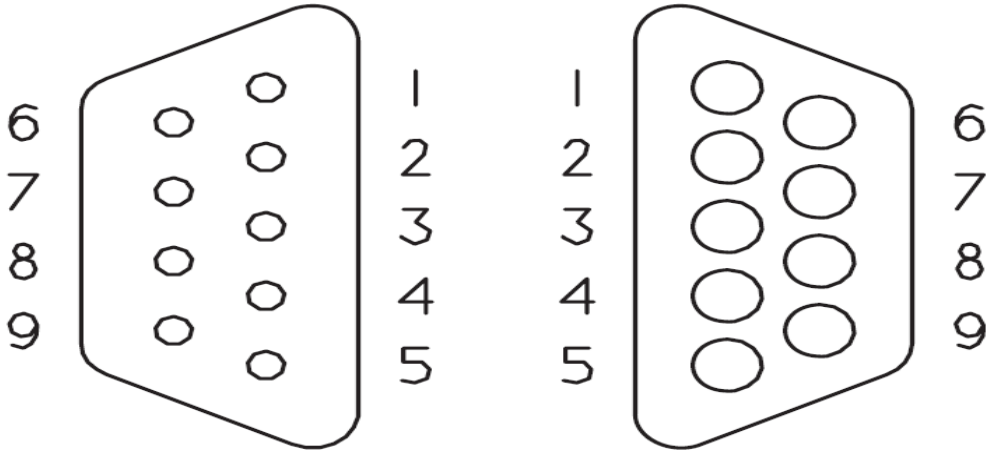
- A line driver is required at the transmitting end to adjust the voltage to the correct level for the communications link (MAX232)

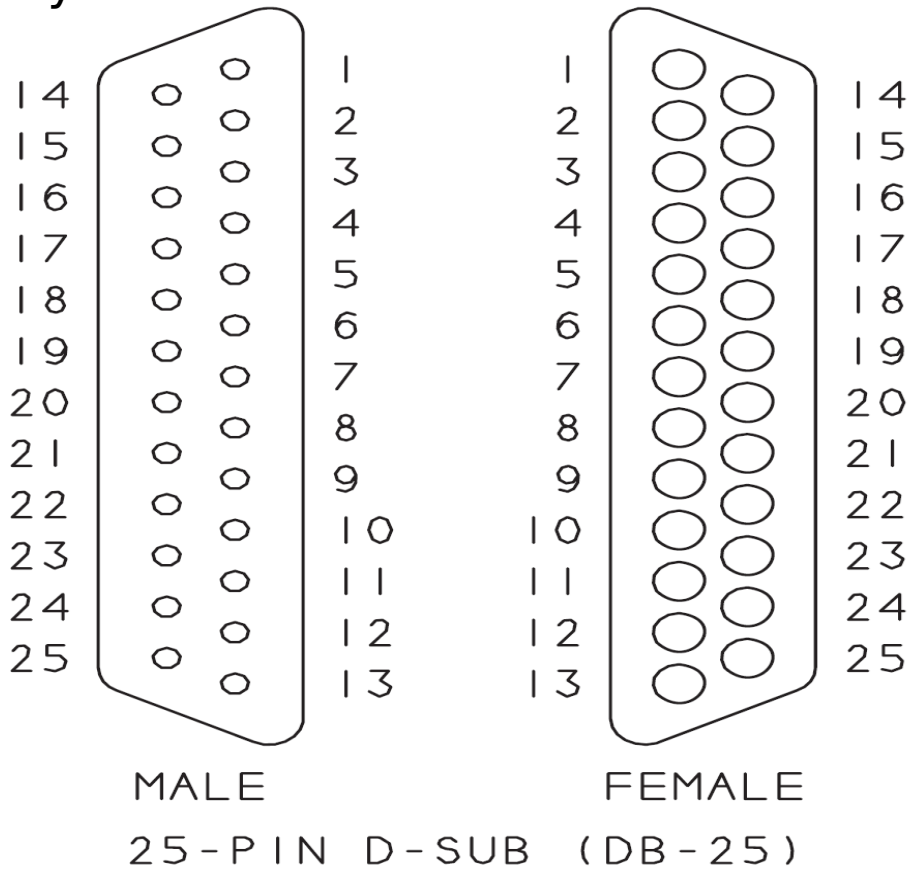
EIA-232 : Electrical signal characteristics



EIA-232 : Mechanical Characteristics

- The DB-25 connector is the de facto standard with revision D
- DB-9 connector has also became an industry standard



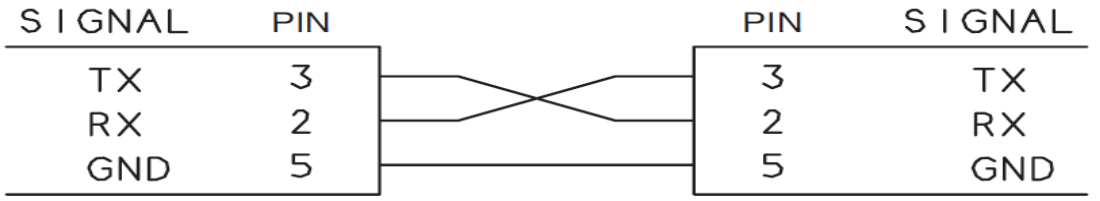


EIA-232 : Mechanical Characteristics

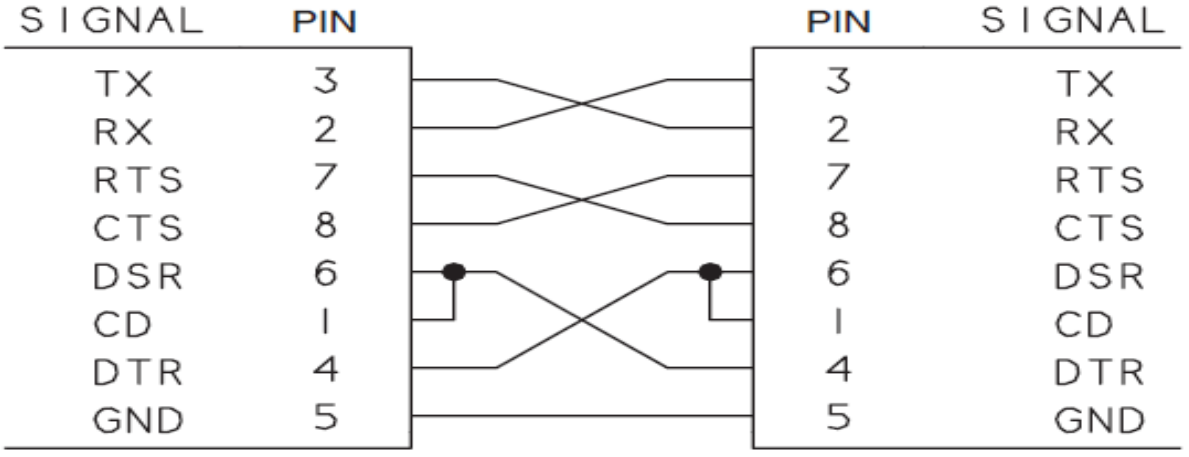
Pin Number (9-pin D-sub)	Pin Number (25-pin D-sub)	Signal	Source	Type	Description
1	8	CD	DCE	control	Carrier detect
2	3	RX	DCE	data	Receive data
3	2	TX	DTE	data	Transmit data
4	20	DTR	DTE	control	Data terminal ready
5	7	SG	–	–	Signal ground
6	6	DSR	DCE	control	Data set ready
7	4	RTS	DTE	control	Request to send
8	5	CTS	DCE	control	Clear to send
9	22	RI	DCE	control	Ring Indicator
–	1, 9-19, 21, 23-25	unused	–	–	–

EIA-232 : Mechanical Characteristics

No Flow Control



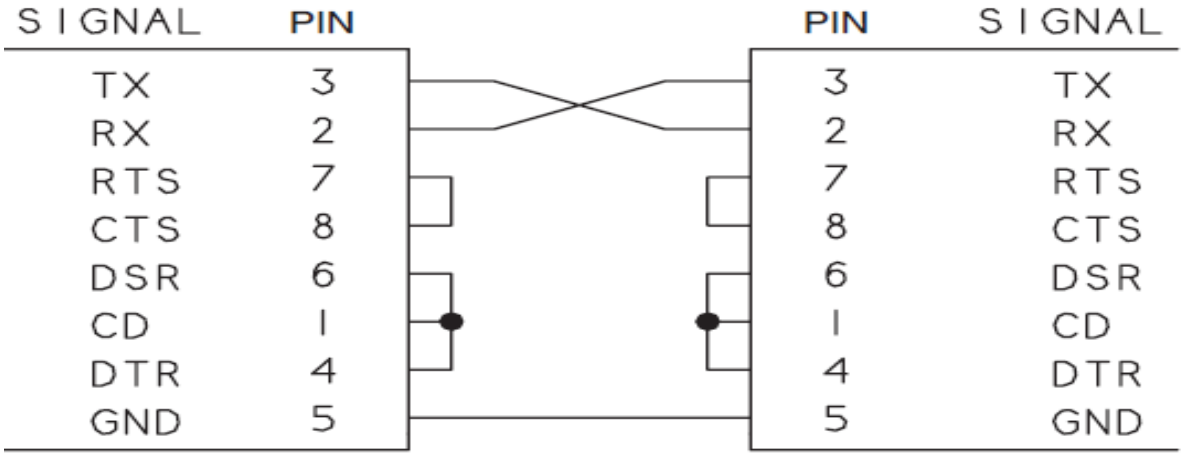
With Flow Control / Hardware Flow Control

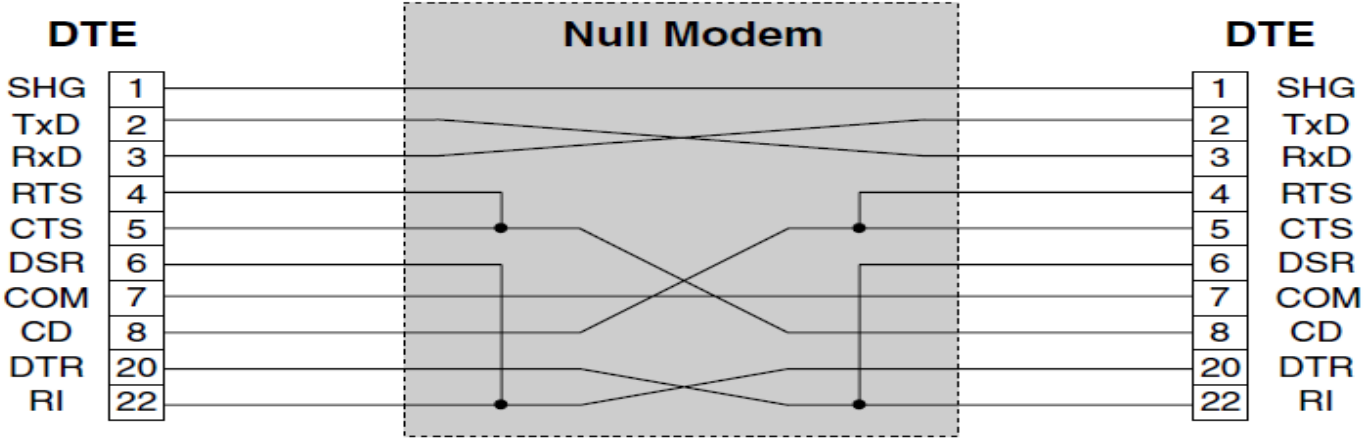


- The handshaking lines the ***enabled*** state means a ***positive voltage*** and the ***disabled*** state means a ***negative voltage***

EIA-232 : Mechanical Characteristics

Loop Back Flow Control
or
Null Modem





EIA-232 : Limitations

- The point-to-point restriction, a severe limitation when several ‘smart’ instruments are used
- The distance limitation of 15 meters (50 feet) end-to-end, too short for most control systems
- The 20 kbps rate, too slow for many applications
- The –3 to –25 V and +3 to +25 V signal levels, not directly compatible with modern standard power supplies.

EIA-232 : Troubleshooting