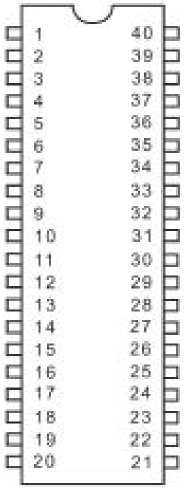
CS122A: Intermediate Embedded and Real Time Operating Systems

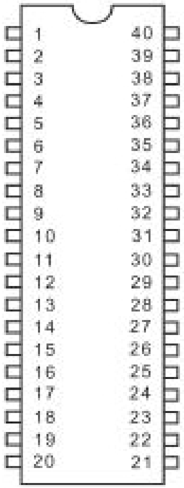
Jeffrey McDaniel

University of California, Riverside

Serial Communication

μC_M μC_S

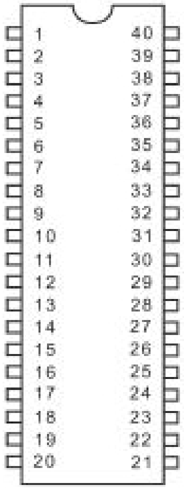


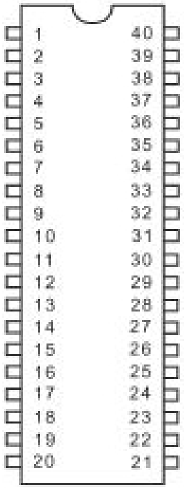


- Microcontrollers need to communicate with each other

Serial Communication

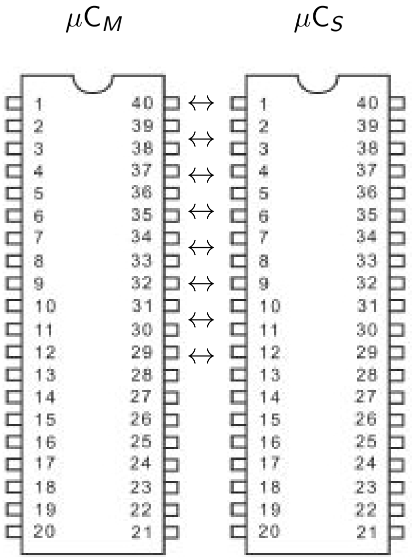
μC_M μC_S





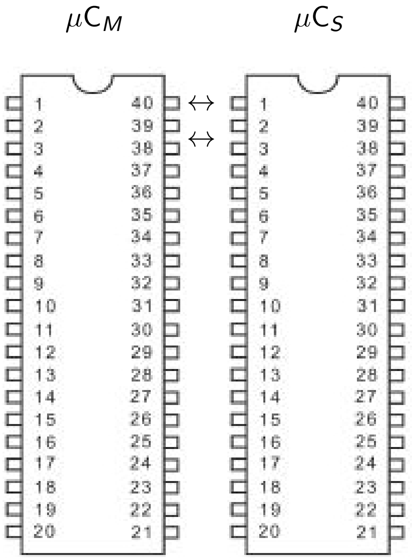
- Microcontrollers need to communicate with each other
- and with other devices

Serial Communication



- Microcontrollers need to communicate with each other
- and with other devices
- Connecting a pin for each bit of data is intractable

Serial Communication



- Microcontrollers need to communicate with each other
- and with other devices
- Connecting a pin for each bit of data is intractable
- **Serial communication** allows us to use fewer pins

Serial Communication

We will be discussing:

- Universal Asynchronous Receiver/Transmitter (UART)

Serial Communication

We will be discussing:

- Universal Asynchronous Receiver/Transmitter (UART)
- Universal Synchronous/Asynchronous Receiver/Transmitter (USART)

Serial Communication

We will be discussing:

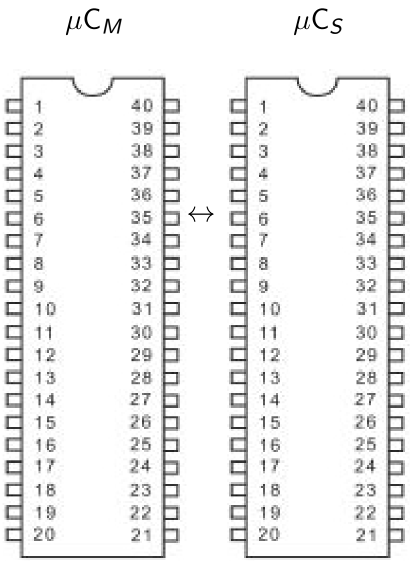
- Universal Asynchronous Receiver/Transmitter (UART)
- Universal Synchronous/Asynchronous Receiver/Transmitter (USART)
- Serial Peripheral Interface (SPI)

Serial Communication

We will be discussing:

- Universal Asynchronous Receiver/Transmitter (UART)
- Universal Synchronous/Asynchronous Receiver/Transmitter (USART)
- Serial Peripheral Interface (SPI)
- Universal Serial Bus (USB)

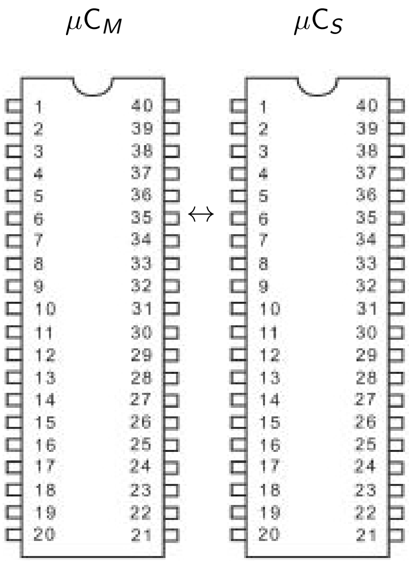
UART Communication



- ATMega's use Universal Asynchronous Receiver/Transmitters (UART)

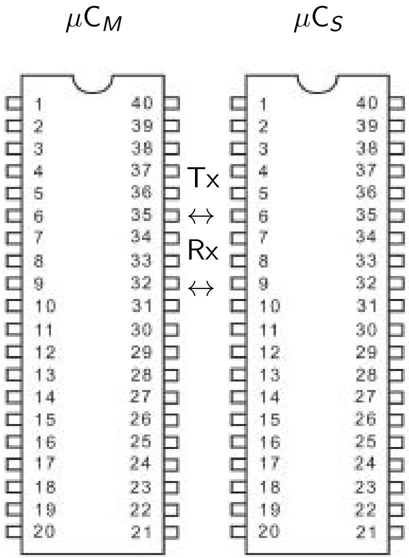
UART Communication

μC_M $\leftrightarrow$ μC_S



- ATMega's use Universal Asynchronous Receiver/Transmitters (UART)
- UART automatically transmits 8 bits of data serially (one bit at a time) over a single pin

UART Communication

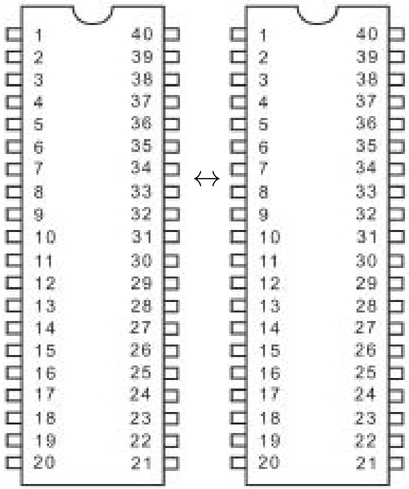


- ATMega's use Universal Asynchronous Receiver/Transmitters (UART)
- UART automatically transmits 8 bits of data serially (one bit at a time) over a single pin
- Transmit (Tx) and Receive (Rx) over two separate pins

UART Communication

μC_M μC_S

T R

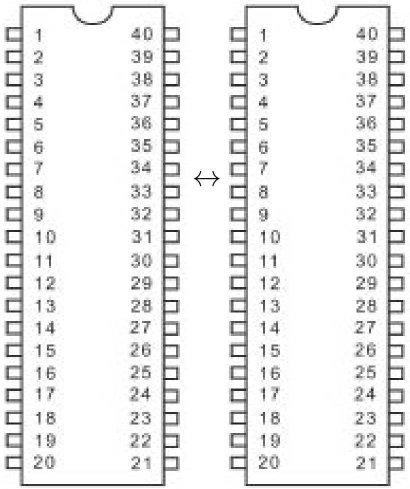


- UART uses a special global variable T to transmit and R to receive

UART Communication

μC_M μC_S

T R



- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission

UART Communication

μC_M

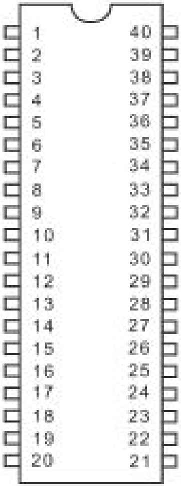
$T = 'a'$

TxReady$= 1$

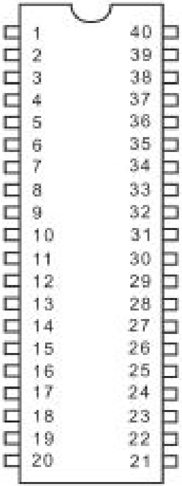
μC_S

R

RxFlag$= 0$

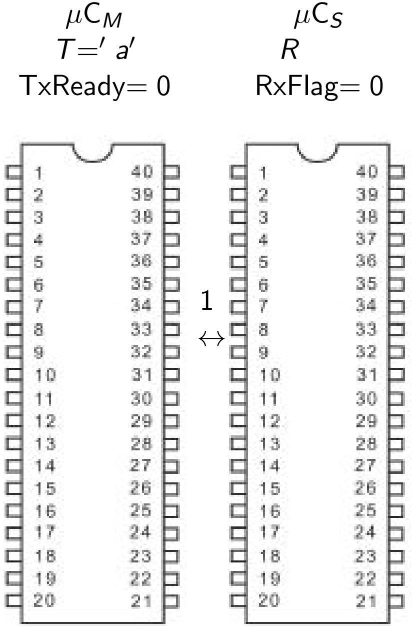


$\leftrightarrow$



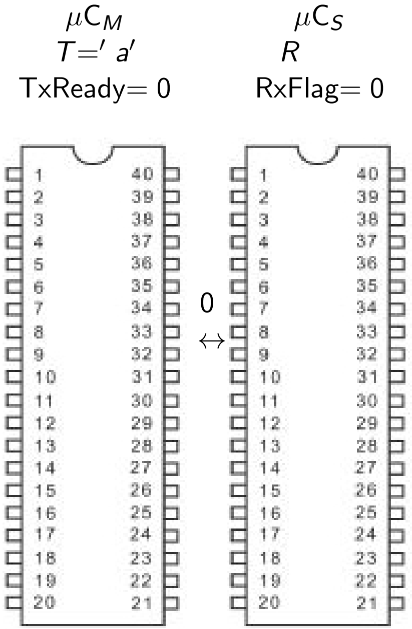
- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively

UART Communication



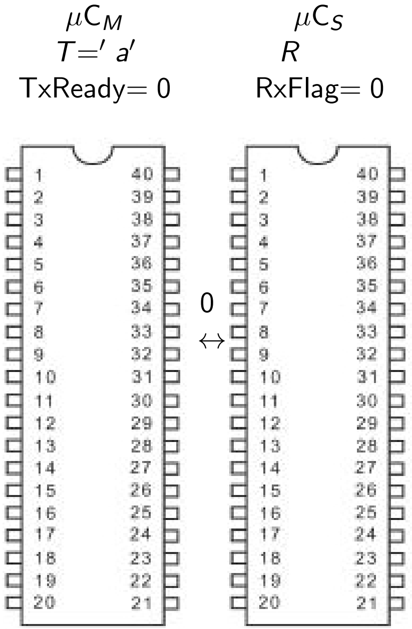
- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively
- T is transmitted a bit at a time (‘a’ is 0b0110 0001)

UART Communication



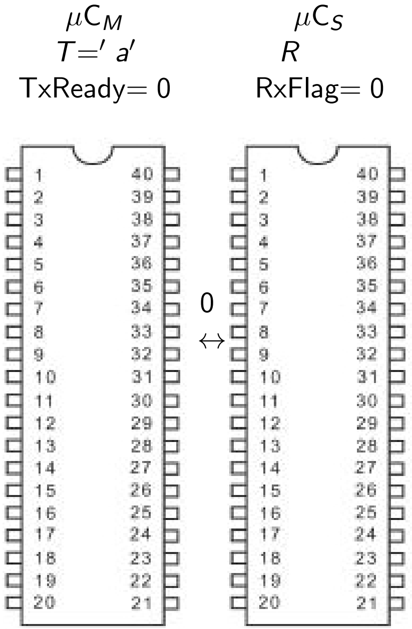
- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively
- T is transmitted a bit at a time ('a' is 0b0110 0001)

UART Communication



- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively
- T is transmitted a bit at a time ('a' is 0b0110 0001)

UART Communication



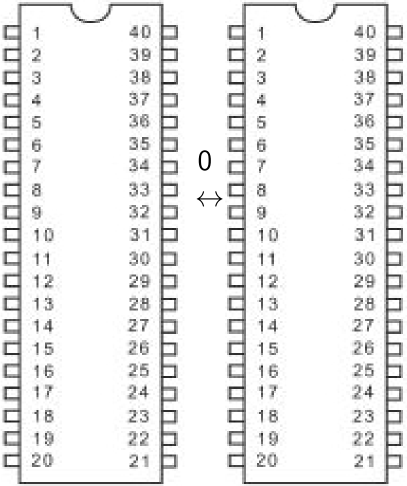
- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively
- T is transmitted a bit at a time ('a' is 0b0110 0001)

UART Communication

μC_M $\quad$ μC_S

$T='a'$ $\quad$ R

TxReady$=0$ $\quad$ RxFlag$=0$



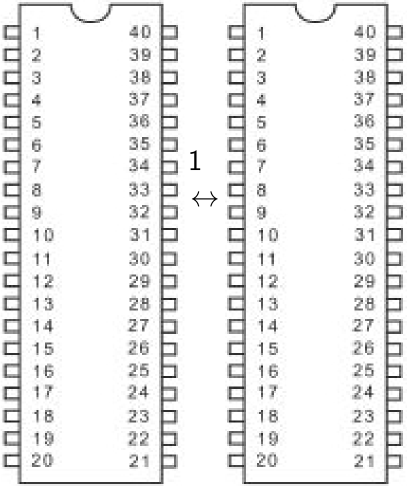
- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively
- T is transmitted a bit at a time ('a' is 0b0110 0001)

UART Communication

μC_M μC_S

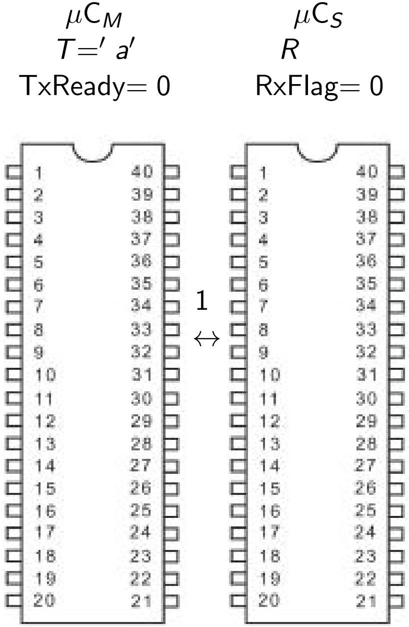
$T = 'a'$ R

TxReady$= 0$ RxFlag$= 0$



- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively
- T is transmitted a bit at a time ('a' is 0b0110 0001)

UART Communication



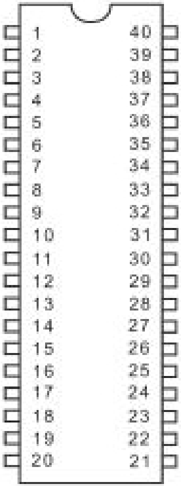
- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively
- T is transmitted a bit at a time ('a' is 0b0110 0001)

UART Communication

μC_M μC_S

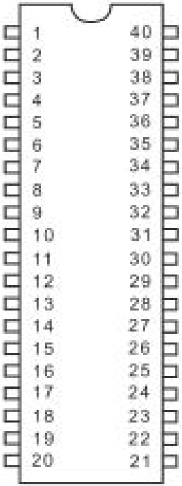
$T = 'a'$ R

TxReady$= 0$ RxFlag$= 0$



0

$\leftrightarrow$

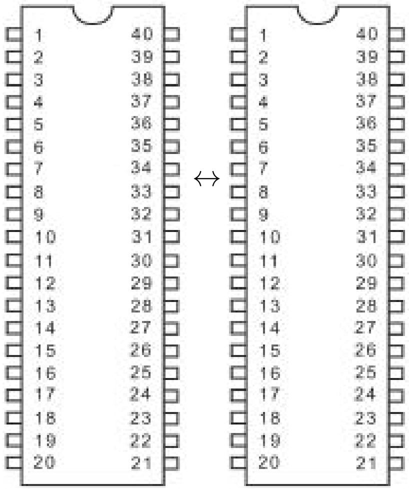


- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively
- T is transmitted a bit at a time ('a' is 0b0110 0001)

UART Communication

μC_M
$T = 'a'$
TxReady$= 1$

μC_S
$R = 'a'$
RxFlag$= 1$



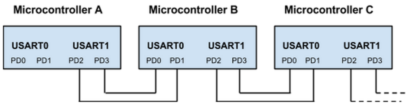
- UART uses a special global variable T to transmit and R to receive
- T should not be changed during transmission
- UART uses flags (TxReady,RxFlag) to indicate it is ready to transmit or has received respectively
- T is transmitted a bit at a time ('a' is 0b0110 0001)

Serial Peripheral Interface (SPI)

- **SPI** is more scalable than USART

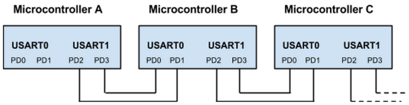
Serial Peripheral Interface (SPI)

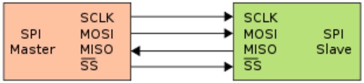
- **SPI** is more scalable than USART
- USART requires 2 wires for each device



Serial Peripheral Interface (SPI)

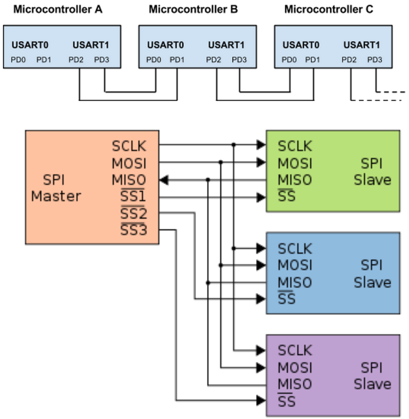
- **SPI** is more scalable than USART
- USART requires 2 wires for each device
- SPI requires 4 for the first





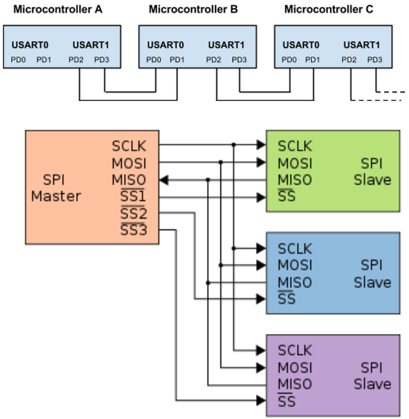
Serial Peripheral Interface (SPI)

- **SPI** is more scalable than USART
- USART requires 2 wires for each device
- SPI requires 4 for the first and 1 for each after



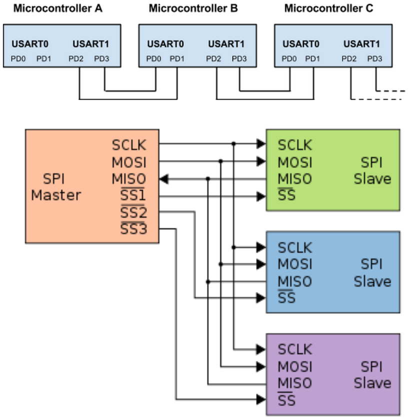
Serial Peripheral Interface (SPI)

- **SPI** is more scalable than USART
- USART requires 2 wires for each device
- SPI requires 4 for the first and 1 for each after
- With 3 peripherals USART and SPI require the same number of wires



Serial Peripheral Interface (SPI)

- **SPI** is more scalable than USART
- USART requires 2 wires for each device
- SPI requires 4 for the first and 1 for each after
- With 3 peripherals USART and SPI require the same number of wires
- With 4 peripherals USART requires more wires than SPI (8:7)



Serial Peripheral Interface (SPI)

SPI terminology:

- Master: Initiates the data frame

Serial Peripheral Interface (SPI)

SPI terminology:

- Master: Initiates the data frame
- Slave: Receives data from master (multiple)

Serial Peripheral Interface (SPI)

SPI terminology:

- Master: Initiates the data frame
- Slave: Receives data from master (multiple)
- SCLK: Serial Clock (output from master)

Serial Peripheral Interface (SPI)

SPI terminology:

- Master: Initiates the data frame
- Slave: Receives data from master (multiple)
- SCLK: Serial Clock (output from master)
- MOSI: Master Output, Slave Input

Serial Peripheral Interface (SPI)

SPI terminology:

- Master: Initiates the data frame
- Slave: Receives data from master (multiple)
- SCLK: Serial Clock (output from master)
- MOSI: Master Output, Slave Input
- MISO: Master Input, Slave Output

Serial Peripheral Interface (SPI)

SPI terminology:

- Master: Initiates the data frame
- Slave: Receives data from master (multiple)
- SCLK: Serial Clock (output from master)
- MOSI: Master Output, Slave Input
- MISO: Master Input, Slave Output
- SS: Slave Select

Serial Peripheral Interface (SPI)

1. Master configures the clock for the specific slave

Serial Peripheral Interface (SPI)

1. Master configures the clock for the specific slave
2. Master transmits a 0 over the desired SS line

Serial Peripheral Interface (SPI)

1. Master configures the clock for the specific slave
2. Master transmits a 0 over the desired SS line
3. During each SPI clock cycle:

Serial Peripheral Interface (SPI)

1. Master configures the clock for the specific slave
2. Master transmits a 0 over the desired SS line
3. During each SPI clock cycle:
    - Master sends a bit on MOSI line

Serial Peripheral Interface (SPI)

1. Master configures the clock for the specific slave
2. Master transmits a 0 over the desired SS line
3. During each SPI clock cycle:
    - Master sends a bit on MOSI line
    - Slave sends a bit on MISO line

Serial Peripheral Interface (SPI)

1. Master configures the clock for the specific slave
2. Master transmits a 0 over the desired SS line
3. During each SPI clock cycle:
    - Master sends a bit on MOSI line
    - Slave sends a bit on MISO line
4. Master stops pulsing CLK

Serial Peripheral Interface (SPI)

1. Master configures the clock for the specific slave
2. Master transmits a 0 over the desired SS line
3. During each SPI clock cycle:
    - Master sends a bit on MOSI line
    - Slave sends a bit on MISO line
4. Master stops pulsing CLK
5. Master deselects SS

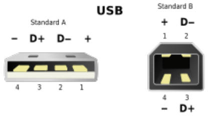
Universal Serial Bus (USB)

- Industry standard developed in mid-1990's

Universal Serial Bus (USB)

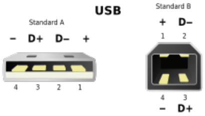
- Industry standard developed in mid-1990's
- Defines cables,

Universal Serial Bus (USB)



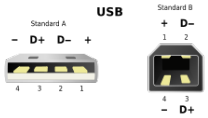
- Industry standard developed in mid-1990's
- Defines cables, connectors and communication protocols

Universal Serial Bus (USB)



- Industry standard developed in mid-1990's
- Defines cables, connectors and communication protocols
- Used for connection, communication, and power supply between computers and electronic devices

Universal Serial Bus (USB)



- Industry standard developed in mid-1990's
- Defines cables, connectors and communication protocols
- Used for connection, communication, and power supply between computers and electronic devices
- Designed to standardize connection of computer peripherals

Universal Serial Bus (USB) - A History

- USB 1.1 Released in 1998 with transfer rate of 1.5Mbit/s (Low-Bandwidth or 12Mbit/s (Full-Bandwidth)

Universal Serial Bus (USB) - A History

- USB 1.1 Released in 1998 with transfer rate of 1.5Mbit/s (Low-Bandwidth or 12Mbit/s (Full-Bandwidth)
- USB 2.0 Released in 2000 adding transfer rates of 35Mbit/s and 280Mbit/s

Universal Serial Bus (USB) - A History

- USB 1.1 Released in 1998 with transfer rate of 1.5Mbit/s (Low-Bandwidth or 12Mbit/s (Full-Bandwidth)
- USB 2.0 Released in 2000 adding transfer rates of 35Mbit/s and 280Mbit/s
- USB 3.0 Released in 2008 adding transfer rate of 4Gbit/s

Universal Serial Bus (USB) - A History

- USB 1.1 Released in 1998 with transfer rate of 1.5Mbit/s (Low-Bandwidth or 12Mbit/s (Full-Bandwidth)
- USB 2.0 Released in 2000 adding transfer rates of 35Mbit/s and 280Mbit/s
- USB 3.0 Released in 2008 adding transfer rate of 4Gbit/s
- USB 3.1 Released in 2013 adding transfer rate of 10Gbit/s