Serial Communication

Outline

- Introduction
- RS232 Standards
- MAX232
- UART
    - Features
    - Framing
    - Pin Details
    - Registers
    - Programming

Serial communication

- In *serial communication*, data is transmitted one bit at a time (Figure a). Thus if an 8-bit word is to be transmitted, the 8 bits are transmitted one at a time in sequence along a cable. This means that a data word has to be separated into its constituent bits for transmission and then reassembled into the word when received.
- In *parallel communication*, all the constituent bits of a word are simultaneously transmitted along parallel cables (Figure b). This allows data to be transmitted over short distances at high speeds.

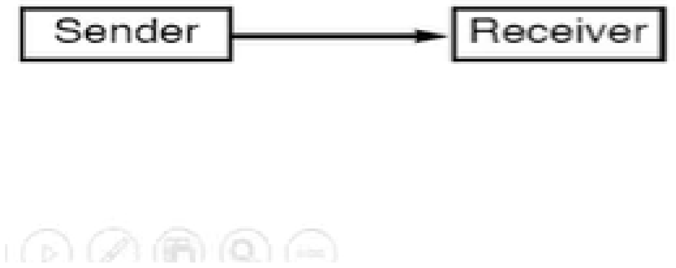


Figure a

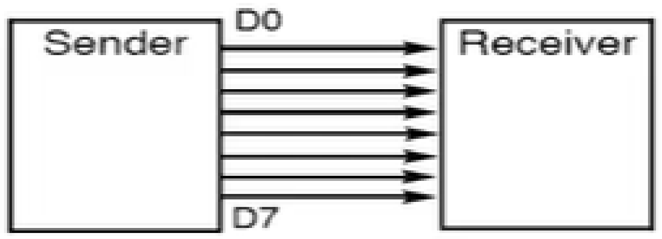


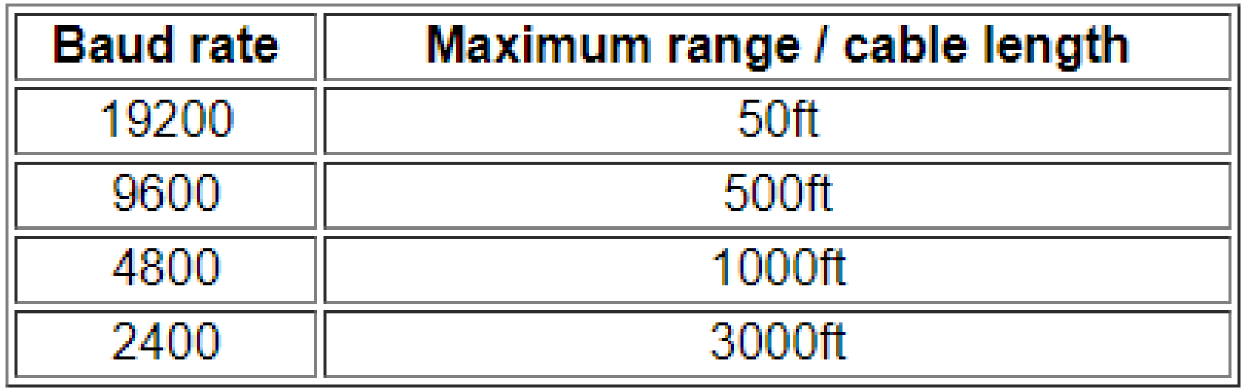
Figure b

Serial communication

- Serial communication has two techniques
  - Synchronous – Transfer block of data at a time
  - Asynchronous – Transfer single byte at a time
- This is achieved by the hardware
  - USART (Universal Synchronous Asynchronous Receiver-Transmitter)
  - UART (Universal Asynchronous Receiver-Transmitter)
- Serial port allows the microcontroller to communicate with devices such as computers, printers, input sensors, and LCDs.
- The total number of bits transmitted per second is called the **baud rate**.
- The reciprocal of the baud rate is the **bit time**, which is the time to send one bit.

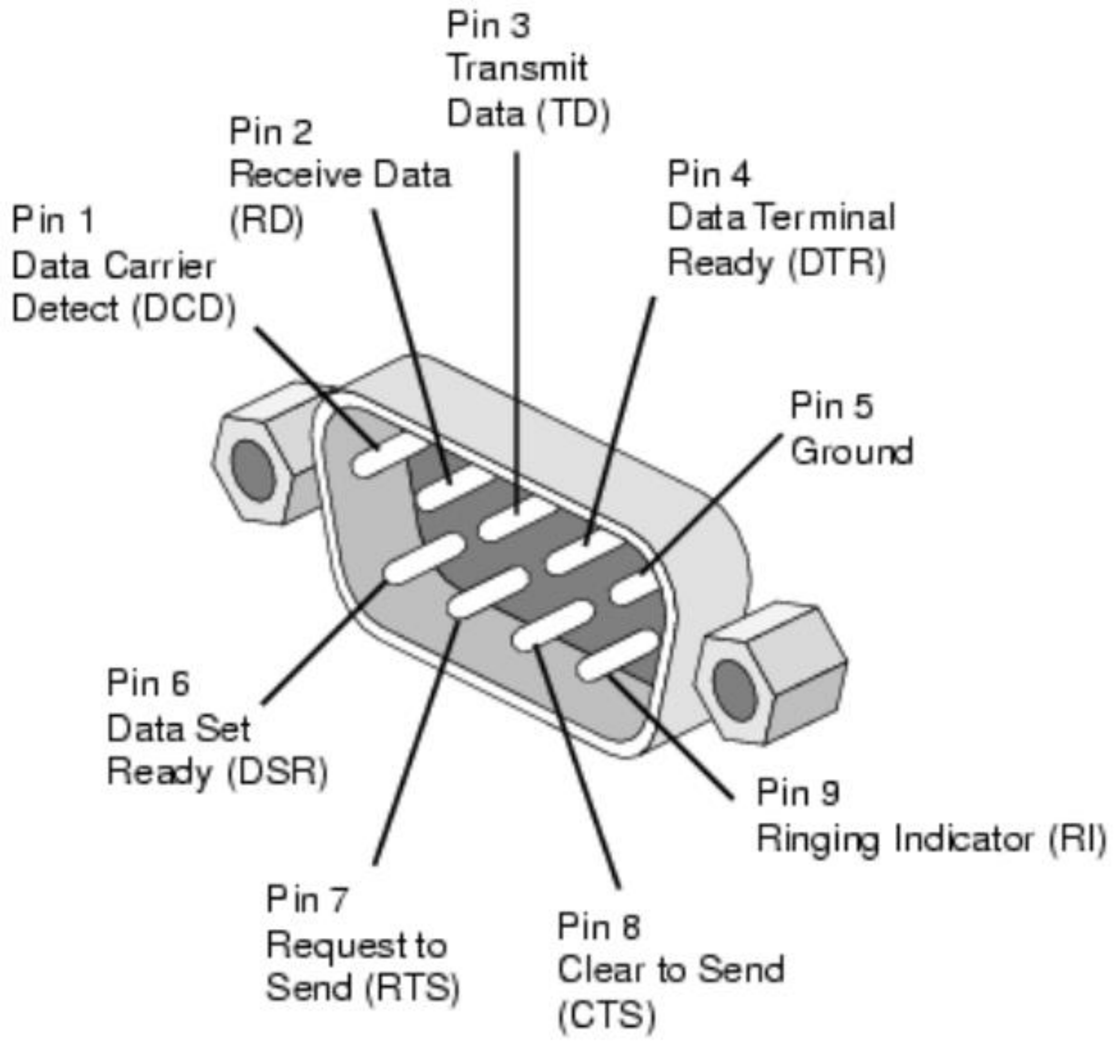
Serial communication: RS232 Standards

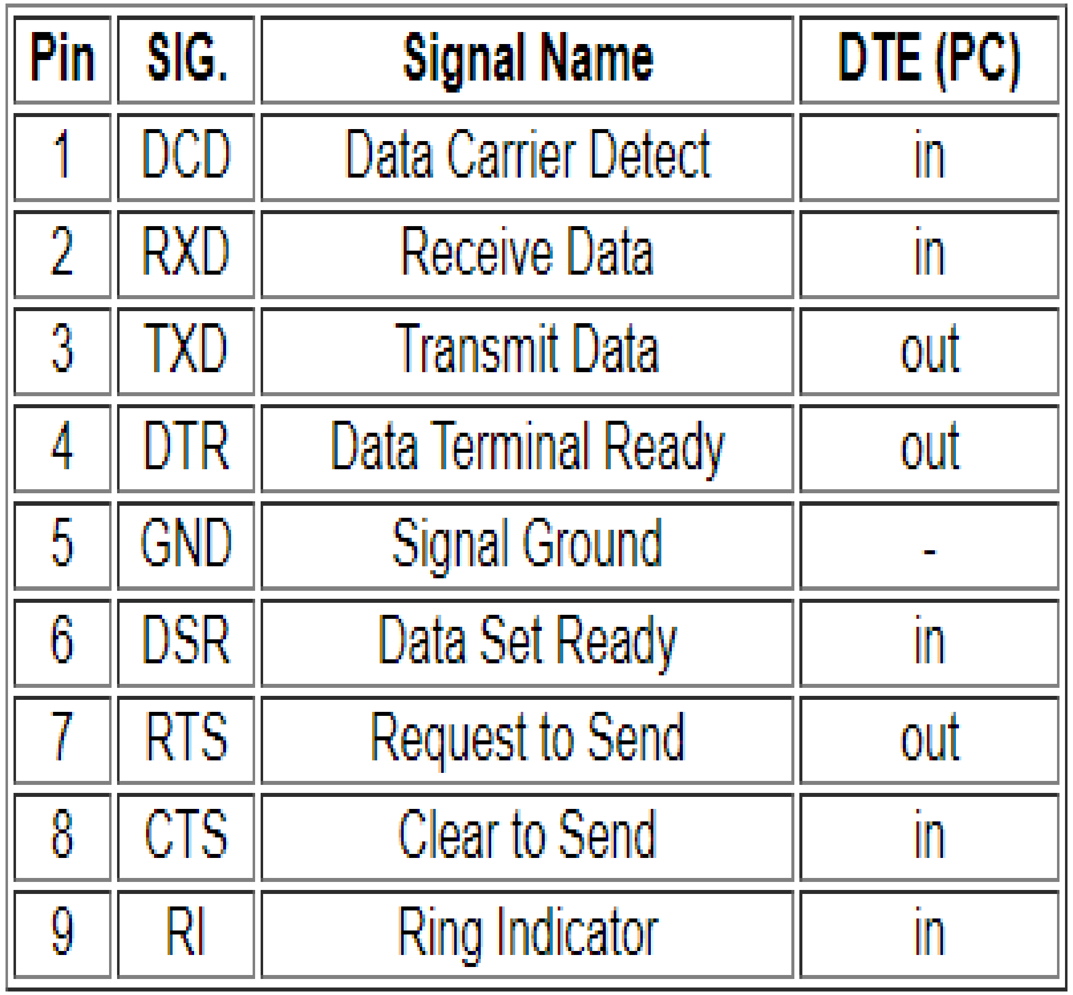
- RS232 standard is the most widely used serial I/O interfacing standard for serial communication
- RS232 cable is used to identify the difference of two signal levels between logic 1 and logic 0.
- The logic 1 is represented by the -12V and logic 0 is represented by +12V.
- Input and output voltage levels are not TTL compatible
- The standard cable length for RS-232 is 50ft, but in practice depends on baud rate



Baud rate	Maximum range / cable length
19200	50ft
9600	500ft
4800	1000ft
2400	3000ft

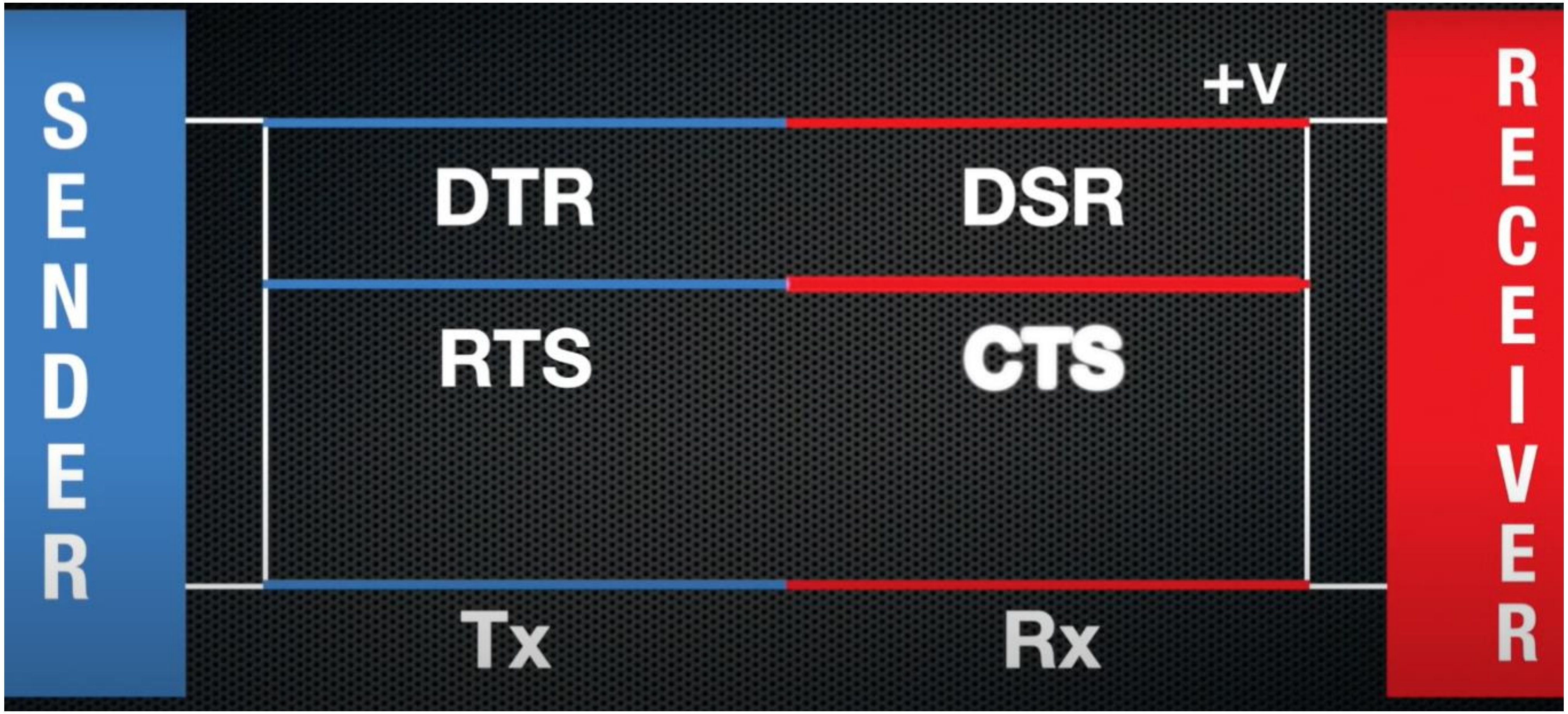
RS232: DB9 Connector





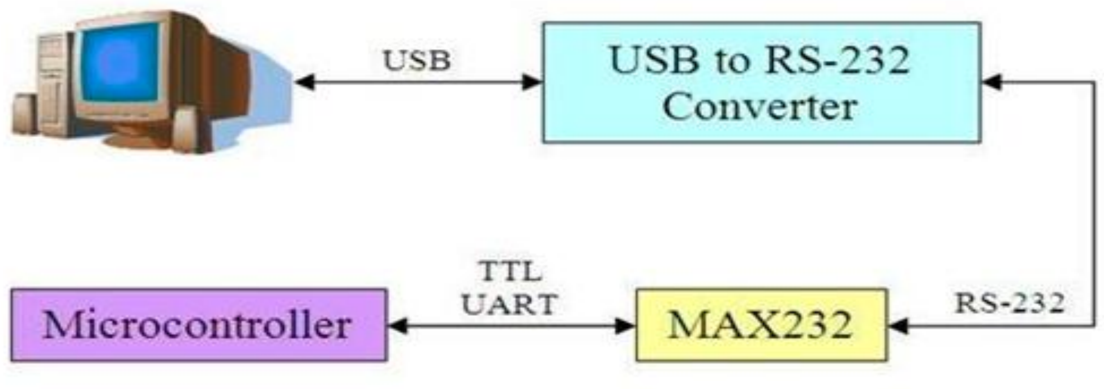
Pin	SIG.	Signal Name	DTE (PC)
1	DCD	Data Carrier Detect	in
2	RXD	Receive Data	in
3	TXD	Transmit Data	out
4	DTR	Data Terminal Ready	out
5	GND	Signal Ground	-
6	DSR	Data Set Ready	in
7	RTS	Request to Send	out
8	CTS	Clear to Send	in
9	RI	Ring Indicator	in

RS232: DB9 Connector



MAX232

- To connect RS232 to a microcontroller, system must use voltage converters such as MAX232 to convert the TTL logic levels to the RS232 voltage levels, and vice versa



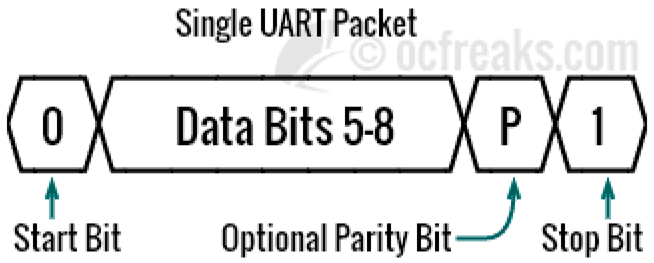
- Max 232 acts as a buffer driver/line driver for the processor.
- Input voltage levels compatible with TTL standard.
- Output voltage levels compatible with RS 232 standard
- Input supply voltage of 5V.

MAX232

- It accepts the standard digital logic values of 0 & 5 volts and converts them to the RS232 standard of +12 & -12 volts.
- Few Microcontrollers have built in serial ports which allow for direct connection with the RS232 serial port of the PC. However many microcontrollers give a 0 to 5V output and require a intermediate buffer circuit to convert the 0 to 5 volts to +12 and -12V required by the RS232 port.
- It is basically a 16 pin IC with the transmitter pins connected to the microcontroller and the port such that the input transmitter pin get TTL input from the Microcontroller and the output transmitter pin supply output to the RS232 port.
- The receiver pins are connected to the RS232 port such that the input receiver pin receive RS232 standard input from the PC port and the output receiver pin supplies the TTL input to the Microcontroller.
- Thus the transmitter takes input from the Microcontroller and gives output to the RS232 port whereas the receiver takes input from the RS232 port and gives output to the Microcontroller.

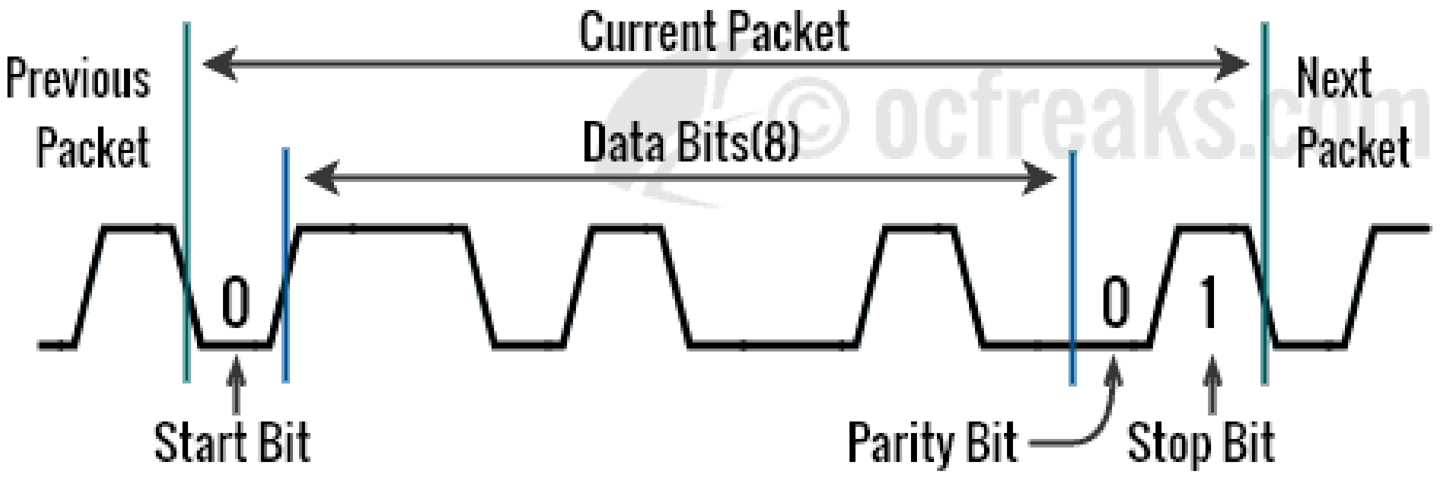
UART

- UART stands for Universal Asynchronous Receiver/Transmitter (UART).
- Asynchronous communication is used for character-oriented transmission
- Each character is placed between start and stop bits and this is called framing.
- A **frame** is the smallest complete unit of serial transmission. There is always only one start bit, and 1 or 2 stop bits. Start bit is always low and stop bit is high.
- UARTs allow us to select 5 to 8 data bits and UART can add even, odd, or no parity bit. LSB is sent out first.
- Figure shows a single frame, which includes a **start bit** (which is 0), data bits (least significant bit first), a parity bit and a **stop bit** (which is 1).



Framing: Example

Here is an example of single Packet/Frame using 8 data bits, even parity and 1 stop bit:



- **Even Parity:** Number of 1s in the transmitted character and the attached parity bit will be even.
- **Odd Parity:** Number of 1s in the transmitted character and the attached parity bit will be odd
- Forced 1 Parity
- Forced 0 Parity

UART

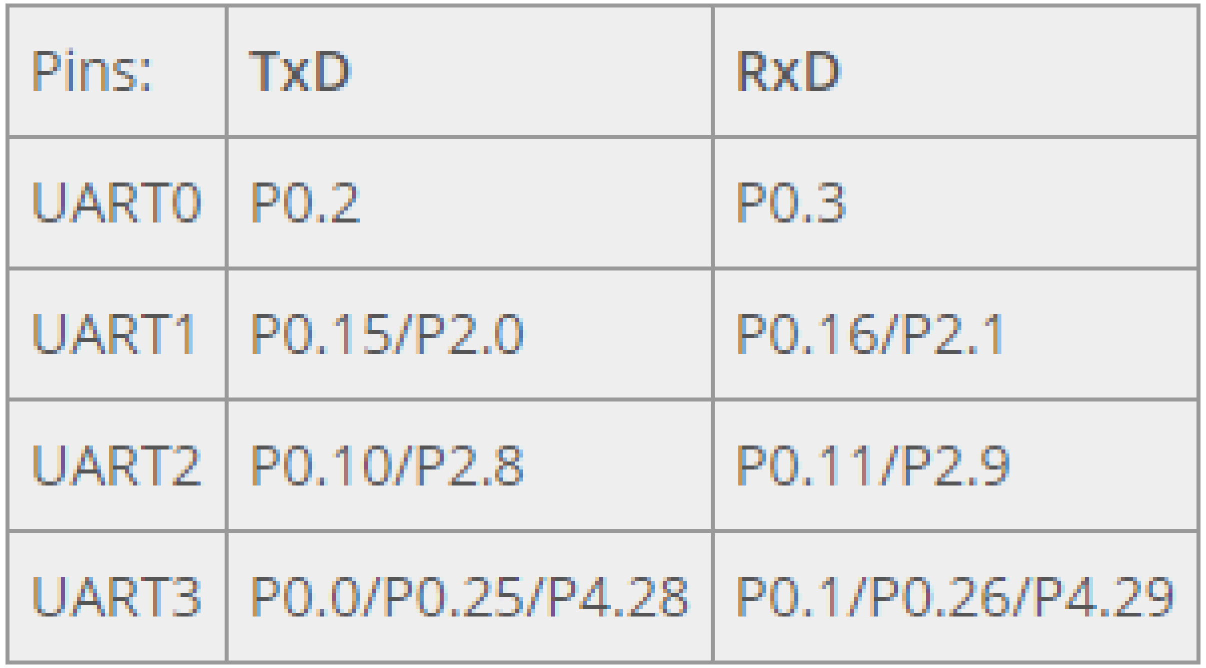
- The LPC 1768 microcontroller consists of 4 UART peripherals. (UART0, UART1, UART2, and UART3).
- Few of the striking features of these peripherals are:
  - Like any other UART peripheral, they can handle data sizes of 5 to 8 bits.
  - They support 16 bytes receive and transmit FIFOs. Which means that they can store 16-bytes of data in a first in first out fashion without overwriting existing data in the FIFO buffer before it gets filled.
  - It has a built-in baud rate generator.

UART

- UART communication basically uses 2 pins for Data transfer and they are
    - TxD (or Tx) – which is the Transmit Data Pin used to send data
    - RxD (or Rx) – which is the Receive Data Pin used to get data

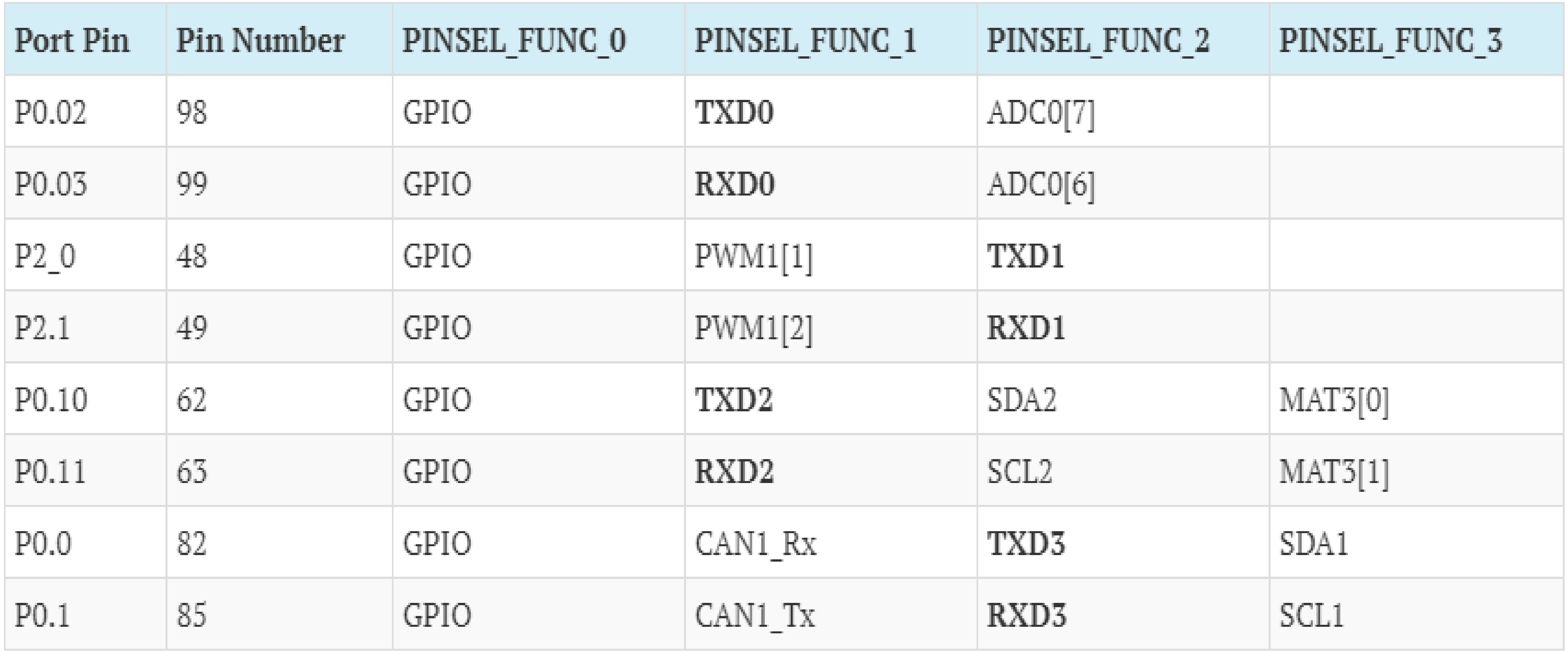
UART

- After reset, UART0 & UART1 are enabled by default.
- TxD and RxD pins for these blocks are mapped on multiple pins.



Pins:	TxD	RxD
UART0	P0.2	P0.3
UART1	P0.15/P2.0	P0.16/P2.1
UART2	P0.10/P2.8	P0.11/P2.9
UART3	P0.0/P0.25/P4.28	P0.1/P0.26/P4.29

UART Pins



Port Pin	Pin Number	PINSEL_FUNC_0	PINSEL_FUNC_1	PINSEL_FUNC_2	PINSEL_FUNC_3
P0.02	98	GPIO	**TXD0**	ADC0[7]	
P0.03	99	GPIO	**RXD0**	ADC0[6]	
P2_0	48	GPIO	PWM1[1]	**TXD1**	
P2.1	49	GPIO	PWM1[2]	**RXD1**	
P0.10	62	GPIO	**TXD2**	SDA2	MAT3[0]
P0.11	63	GPIO	**RXD2**	SCL2	MAT3[1]
P0.0	82	GPIO	CAN1_Rx	**TXD3**	SDA1
P0.1	85	GPIO	CAN1_Tx	**RXD3**	SCL1

UART Baud Rate Calculation

- The main formula for calculating baud rate is given as:

$$\text{Baud Rate} = \frac{\text{PCLK in Hertz}}{16 \times (256 \times \text{DLM} + \text{DLL}) \times (1 + \text{DIVADDVAL}/\text{MULVAL})}$$

Where DIVADDVAL & MULVAL are part of “Fractional Rate Divider” or “Baud-Prescaler”.

- This “Fractional Divider” is only active when DIVADDVAL > 0.
- This can be further simplified to :

$$\text{Baud Rate} = \frac{\text{PCLK in Hertz}}{16 \times (DL_{est}) \times FR_{est}}$$

$$\text{Where } DL_{est} = 256 \times \text{DLM} + \text{DLL}$$

$$\text{and } FR_{est} = \frac{\text{MULVAL} + \text{DIVADDVAL}}{\text{MULVAL}}$$

UART Baud Rate Calculation

With following conditions strictly applied:

- 0 < MULVAL <= 15
- 0 <= DIVADDVAL <= 14 - if DIVADDVAL > 0 & DLM = 0 then, DLL must be >= 2
- DIVADDVAL < MULVAL

- Where PCLK is the Peripheral Clock value in Hz, DLM and DLL are the divisor registers which we saw earlier and finally DIVADDVAL and MULVAL are part of the Fractional baud rate generator register.

Some of the standard Baud rates that can be used are: 2400, 4800, 7200, 9600, 14400, 19200, 28800, 38400, 57600, 115200, 230400, etc..

Refer to Page 315 of your user manual for calculating the DIVADDVAL, MULVAL, DLL and DLM to obtain the required baud rate.

UART Registers

- Data Related Registers
- Baud Rate Setup Related Registers
- Control Registers
- Status Register
- Interrupt Related Registers

UART Registers

- All blocks contain 2 registers each, for data access and assembly as given below:
  - **Tx has THR (Transmit Holding Register) and TSR (Transmit Shift Register)** – When Tx Data is written to THR, it is then transferred to TSR which assembles the Transmit data.
  - **Rx has RSR (Receive Shift Register) and RBR (Receive Buffer Register)** – When Data is Received at the Rx Pin, it is first assembled in RSR and then transferred into Rx FIFO which can be then accessed using RBR.

UART Data Related Registers

- **UxRBR – Receiver Buffer Register:** This register contains the top most byte (8-bit data chunk) in the Rx FIFO i.e the **oldest** received data in FIFO. Before reading from UxRBR, the **DLAB** (Divisor Latch Access) bit in **UxLCR** register must be **0**. The right approach for fetching the valid pair of received byte and its status bits is first to read the content of PE, FE and BI bits in the UxLSR register, and then to read a byte from the UxRBR.
- **UxTHR – Transmit Holding Register:** UxTHR contains the top most byte in Tx FIFO and in this case its the **newest** (latest) transmitted data. As in the case with UxRBR, we must set DLAB=0 to access UxTHR for write operation.

UART Baud Rate Setup related registers

- **UxDLL and UxDLM – Divisor Latch registers:**
  - Both of them hold 8-bit values.
  - These registers together form a 16-bit divisor value which is used in baud rate generation.
  - UxDLM holds the upper 8-bits and U0DLL holds the lower 8-bits and the formation is “**[U0DLM:U0DLL]**“. Since these form a divisor value and division by zero is invalid, the starting value for U0DLL is 0x01 (and not 0x00).
- **In order to access and use these registers properly, DLAB bit in UxLCR must be first set to 1**.

UART Baud Rate Setup related registers

- **UxFDR – Fractional Divider Register :** It is used to set the prescale value for baud rate generation. The input clock is the PCLK and output is the desired clock defined by this register. This register actually holds two different 4-bit values (a divisor and a multiplier) for prescaling which are:
  - **Bit [3 to 0] – DIVADDVAL:** This is the prescale divisor value. If this value if 0, then fractional baud rate generator wont have any effect on Uart Baud rate.
  - **Bit [7 to 4] – MULVAL:** This is prescale multiplier value. Even if fractional baud rate generator is not used, the value in this register must be more than or equal to 1, else UART0 will not operate properly.
  - Other Bits reserved.
- Remark from User-manual: **If the fractional divider is active (DIVADDVAL > 0) and DLM = 0, the value of the DLL register must be 2 or greater!**

UART Control and Status Registers

- **UxFCR – FIFO Control Register:** Used to control Rx/Tx FIFO operations.
  - **Bit 0 – FIFO Enable:** 1 to Enable both Rx and Tx FIFOs and 0 to disable.
  - **Bit 1 – Rx FIFO Reset:** Writing a 1 will clear and reset Rx FIFO.
  - **Bit 2 – Tx FIFO Reset:** Writing a 1 will clear and reset Tx FIFO.
  - **Bit 3 – DMA Mode**
  - **Bits [7 to 6]**: **Rx-TRIGGER** Used to determine that how many Rx FIFO characters must be written before an interrupt is activated.
    - 00-- Trigger level 0 (1 character or 0x01)
    - 01-- Trigger level 1 (4 characters or 0x04)
    - 10-- Trigger level 2 (8 characters or 0x08)
    - 11-- Trigger level 3 (14 characters or 0x0E)
  - Others bits are reserved.

UART Control and Status Registers

- **UxLCR – Line Control Register :** Used to configure the UART block (i.e the data format used in transmission).
  - **Bit [1 to 0] – Word Length Select:** Used to select the length of an individual data chunk. [00] for 5 bit character length. Similarly [01] , [10] , [11] for 6, 7, and 8 bit character lengths respectively.
  - **Bit 2 – Stop bit select:** 0 for using 1 stop bit and 1 for using 2 stop bits.
  - **Bit 3 – Parity Enable:** 0 to disabled Parity generation & checking and 1 to enable it.
  - **Bit [5 to 4] – Parity Select:** [00] to Odd-parity , [01] for Even-parity , [10] for forced “1”(Mark) parity and [11] for forced “0”(Space) parity.
  - **Bit 6 – Break Control:** 0 to disable break transmission and 1 to enable it. TxD pin will be forced to logic 0 when this bit is 1.
  - **Bit 7 – Divisior Latch Access bit:** 0 to disable access to divisor latches and 1 to enable access.

UART Control and Status Registers

- **UxTER – Transmit Enable Register:**
  - This register is used to enable UART transmission.
  - When bit-7 (i.e TXEN) is set to 1, Tx block will be enabled and will keep on transmitting data as soon as its ready. If bit-7 is set to 0, then Tx will stop transmission. Other bits are reserved.

UART Control and Status Registers

- **UxLSR – Line Status Register:** used to read the status of Rx and Tx blocks. Bits 1 to 4 get cleared after reading UxLSR.
  - **Bit 0 – Receiver Data Ready (RDR):** 0 means UxRBR is empty (i.e Rx FIFO is empty) and 1 means UxRBR contains valid data.
  - **Bit 1 – Overrun Error (OE):** 1 means Overrun has occurred, 0 otherwise. Overrun is the condition when RSR (Receive Shift Register) has new character assembled, but the RBR FIFO is full and the new assembled character was eventually lost since no data is written into FIFO when its full.
  - **Bit 2 – Parity Error (PE):** 1 means a parity error has occurred else not. When the value of the parity bit in the received character is in wrong state then a parity error occurs.
  - **Bit 3 – Framing Error (FE):** 1 means that a framing error has taken place else not. Framing error occurs when the stop bit of a received character is zero.

UART Control and Status Registers

- **UxLSR – Line Status Register:**
  - **Bit 4 – Break Interrupt:** 1 means that it has occurred else not. A Break Interrupt occurs when the RxD line is pulled low (i.e all 0s) i.e held in spacing state for 1 full character after which Rx Block goes into Idle state. Rx Block gets back to active state when RxD pin is pulled high (i.e all 1s) i.e held in marking state for 1 full character.
  - **Bit 5 – Transmit Holding Register Empty (THRE):** 0 means UxTHR has valid data and 1 means its empty.
  - **Bit 6 – Transmitter Empty (TEMT):** 0 means UxTHR and/or UxRSR has valid data and 1 means that both UxTHR and UxRSR are empty.
  - **Bit 7 – Error in RX FIFO (RXFE):** 0 means that UxRBR has no Rx Errors or Rx FIFO is disabled (i.e 0th bit in U0FCR is 0) and 1 means that UxRBR has at-least one error. **Note:** This bit is cleared only if UxLSR is read and there are no other subsequent errors in Rx FIFO, else this bit will stay 1

UART Interrupt Related Registers

- Interrupt Enable Register
- Interrupt Identification Register

UART Programming

- **Steps for configuring UART**
  1. **Power**: In the PCONP register, set bits PCUART0/2/3.

     Remark: On reset, UART0 is enabled (PCUART0 = 1), and UART2/3 are disabled (PCUART2/3 = 0).
  2. **Peripheral clock**: In the PCLKSEL0 register, select PCLK_UART0; in the PCLKSEL1 register, select PCLK_UART2/3.
  3. **LCR**: Configure for 8-data bits, 1 stop bit, no parity
  4. **Baud rate**: In register U0/2/3LCR, set bit DLAB =1. This enables access to registers DLL and DLM for setting the baud rate. Also, if needed, set the fractional baud rate in the fractional divider register.
  5. **UART FIFO**: Use bit FIFO enable (bit 0) in register U0/2/3FCR to enable FIFO.
  6. **Pins**: Select UART pins through the PINSEL registers
  7. **Clear DLAB**: Finally clear DLAB bit in LCR to disable the access to DLM and DLL registers

UART Programming: Initialization

```
void initUART0(void)
{           /*Assuming CCLK = 12Mhz, PCLK = 3Mhz and Req. Baud Rate=9600*/
            //LPC_SC->PCONP |= 1<<3; //Power up UART0 block. By Default it is enabled after RESET.
            LPC_UART0->LCR = 3 | 1<<7 ; // 8 bits, no Parity, 1 Stop bit & DLAB set to 1
            LPC_UART0->DLL = 0x0D;
            LPC_UART0->DLM = 0;
            LPC_UART0->FDR = (2<<4) | 1; // MULVAL=2(bits - 7:4), DIVADDVAL=1(bits - 3:0)
            LPC_UART0->FCR |= 1<<0 | 1<<1 | 1<<2 ;
            LPC_PINCON->PINSEL0 |= (1<<4) | (1<<6); //Select TXD0 and RXD0 function for P0.2 & P0.3
            LPC_UART0->LCR &= 0<<7; //Now since we have applied DLL and DLM, lock those values by disabling DLAB
}
```

UART Programming

- **Write Data to UART0**

```
void U0Write(char txData)
{
    while(!(LPC_UART0->LSR & 1<<5)); //wait until THR is empty
    //now we can write to Tx FIFO
    LPC_UART0->THR = txData;
}
```

- **Read Data from UART0**

```
char U0Read(void)
{
    while(!(LPC_UART0->LSR & 1<<0)); //wait until any data arrives in Rx FIFO
    return LPC_UART0->RBR;
}
```

References

- https://www.elprocus.com/max232/
- https://exploreembedded.com/wiki/LPC1768:_UART_Programming
- http://www.ocfreaks.com/lpc1768-uart-programming-tutorial/
- http://www.ocfreaks.com/uart-tutorial-basics/
- https://www.db9-pinout.com/
- UM10360_LPC176x: Chapter 14