# UART (Universal Asynch Tx & Rx) Communication Protocol

- A serial comms protocol that allows device to exchange data without a shared clk signal
- It uses min. 2 wires (Tx and Rx) and requires both devices to agree on param. like Baud rate (speed) and data format forehand
- UART Tx (Transmitter) :-

⇒ Ports :- i/p - clk, rst, start, data[7:0]
o/p - tx (serial line, idle high)

⇒ Internal state :-

- Sending (1-bit) : whether a Tx is in progress
- Count (4-bit) : 0...9 indicates which bit is driven

↳ 0 → start bit (0) 1...8 → data bits data[0]... data[7] (LSB 1st)
9 → stop bit (1)

⇒ Reset Behaviour : on reset tx = 1 (idle high)
sending = 0, count = 0

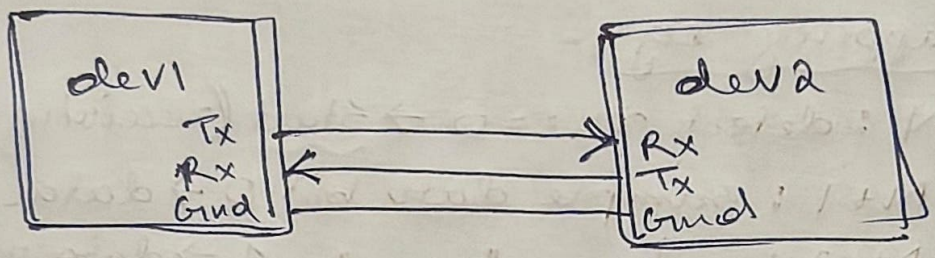
⇒ Start & Transmission :

- on Rising clk
  → If start && !sending - begin new transfer (sending = 1, count = 0)

→ If sending :
- Drive tx according to count
- Increment count
- If count == 9 after increment, deassert sending ( Tx finish)

⇒ Cycle Mapping :-

cycle 0 : start bit (tx=0)
cycle 1 → 8 : data[0] → data[7]
cycle 9 : stop bit (tx=1) ⇒ sending clr




• UART Rx (Receiver)

⇒ ports :- i/p - clk, rst, rx (serial i/p)
- o/p - data_out [7:0], done

⇒ Internal state :-

• receiving (1-bit) : true while frame is being sampled.

• count (4-bit) : 0...9 progress through frame

→ detect start - receiving = 1, count = 0

→ on subsequent clk

▶ count == 1...8 ⇒ sample and store bits into data_out [0..7]

▶ count == 9 ⇒ interpret as stop bit, set done = 1, receiving = 0

⇒ Start detection :-

when not receiving (rx==0) at clk edge, code treats it as start bit and begins sampling from next clock

⇒ Sampling discipline used

- The design samples rx once per bit at the rising clk edge
- The code assumes Tx & Rx share the same clk.

⇒ Cycle Mapping eg :-

clk egde N : detect rx==0 → start (receiving=1, count=0)
clk egde N+1 : sample data bit 0 → data_out[0]=rx
" " N+2 : " " " 1 → data_out[1]=rx
⋮ ⋮ ⋮ ⋮ ⋮ ⋮
clk edge N+8 : sample data bit 7 → data_out[7]=rx
clk edge N+9 : sample data STOP (done=1, receiving=0)

- waveform / timing analysis

Assume data = 0xA5 = 10100101 (MSB..LSB)
LSB-first stream 10100101 ⇒ if Ab = 10100110 ↳ LSB 1st 01100101
d7 —— d1 d0

| Cycle | Tx | Rx samples |
|---|---|---|
| 0 | start=0 | detect start (Rx=0) |
| 1 | data[0]=1 | sample bit-0 → 1 |
| 2 | data[1]=0 | sample bit-1 → 0 |
| 3 | data[2]=1 | sample bit-2 → 1 |
| 4 | data[3]=0 | sample bit-3 → 0 |
| 5 | data[4]=0 | sample bit-4 → 0 |
| 6 | data[5]=1 | sample bit-5 → 1 |
| 7 | data[6]=0 | sample bit-6 → 0 |

data[x] = dx

8 data[7] = 1 sample bit-7 → 1

9 STOP sample stop → done = 1

- Practical Add-up

(1) Baud Rate generator :-

use baud-tick to advance count and sampling rather than clk

(2) Oversampling on Rx :-

when rx sampled eg 8x and 16x baud, detect start when rx = low, then sample data bits at mid bit.

(3) synchronize rx :-

use 2-stage synch flip-flop before rx to avoid Metastability.

(4) Framing and parity :-

Add checks for stop bit = 1

optional add of parity gen/check

(5) FIFO Buffer :-

Add small FIFO on Tx/Rx sides so CPU can push/pull bytes Asynch without missing data

(6) Non-blocking updates in code

(7) done handshake :-

instead of using one-cycle pulse, consider valid+ready handshake to avoid missed reads.