UARTs

UART: Universal Asynchronous Receiver Transmitter

First/Original “Serial Port”; dates from 1960s

UARTs drove PC-based serial communications ports for 40+ years

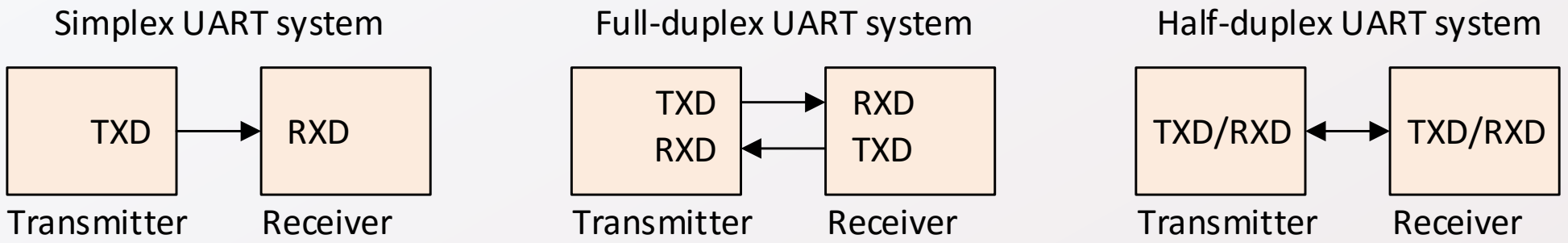
Most serial ports used RS-232, but that’s not a part of the UART spec (RS-232 was developed for modems, and UARTs already existed)

Pin	Signal	Name	Direction	Description
1	CD	Carrier Detect	DTE ← DCE	Indicates Modem connected to phone line
2	RXD	Receive Data	DTE ← DCE	Data signal from peripheral to host (terminal)
3	TXD	Transmit Data	DTE → DCE	Data signal from host (terminal) to peripheral
4	DTR	Data Terminal Ready	DTE → DCE	Indicates terminal is enabled
5	SG	Signal Ground		
6	DSR	Data Set Ready	DTE ← DCE	Indicates peripheral is ready
7	RTS	Request to Send	DTE → DCE	Handshake signal indicating terminal data ready
8	CTS	Clear to Send	DTE ← DCE	Handshake signal indicating peripheral data ready
9	RI	Ring Indicator	DTE ← DCE	Indicates modem phone line is ringing

UART: Universal Asynchronous Receiver Transmitter

UARTs use unidirectional, asynchronous signals – one signal to transmit (TX), one to receive (RX)

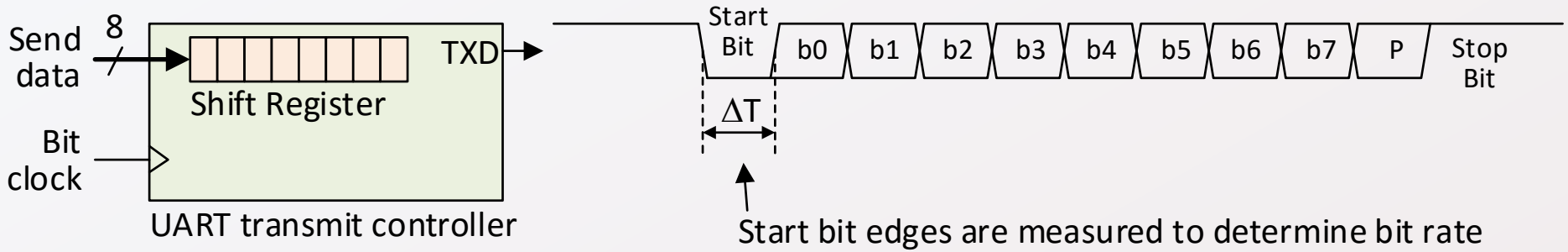
A UART device is required on both side of the transmission system

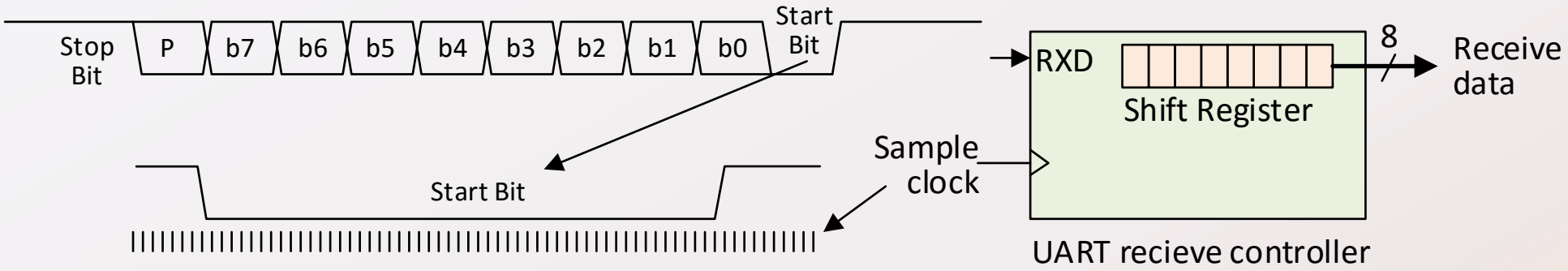


Asynchronous Communications

"Asynchronous" comes from fact that no timing signal is used

Receiver must know transmitter bit-clock frequency, or extract it from data signal

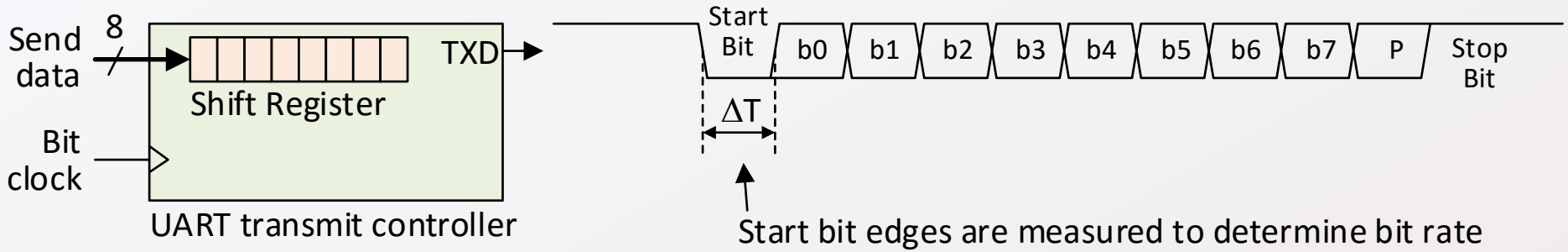


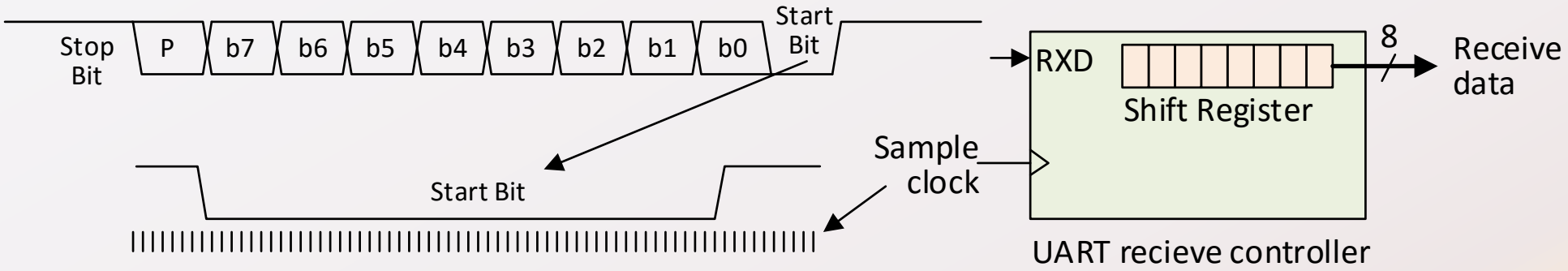


Asynchronous Communications

Data line held high when TXD idle (dates from telegraph days)

Data packets are sporadic; TXD only driven when packet being sent (line is mostly idle)



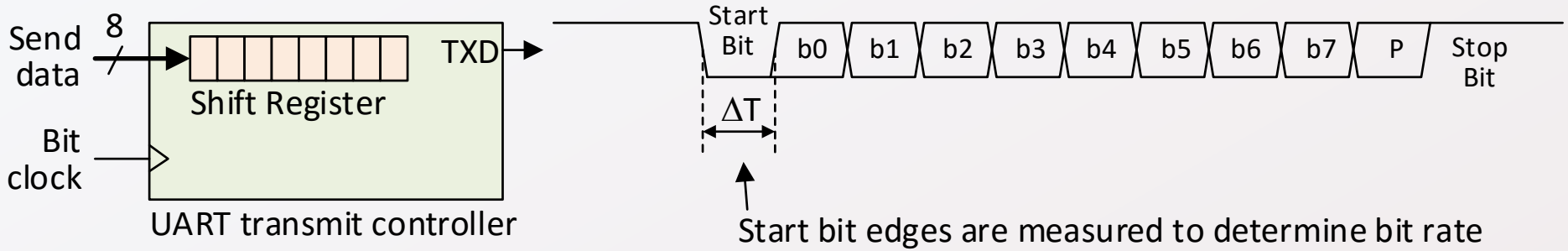


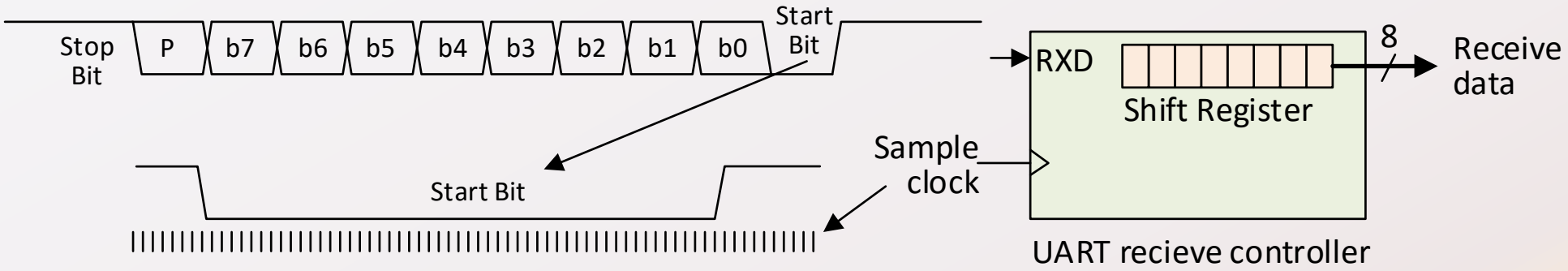
Asynchronous Communications

Parallel data is written to UART register

Transmitter sends data bits out in sequence one at a time, LSB first

Transmitter adds start bit, then data bits, then optional parity bit, then one or two stop bits (why does this matter?)



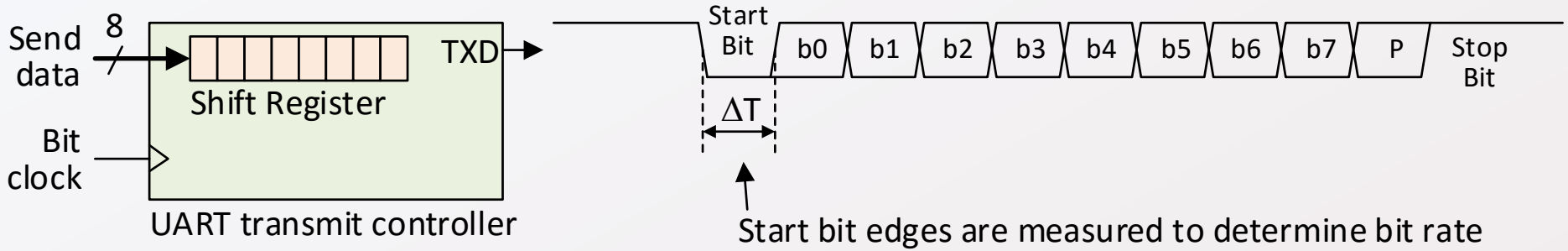


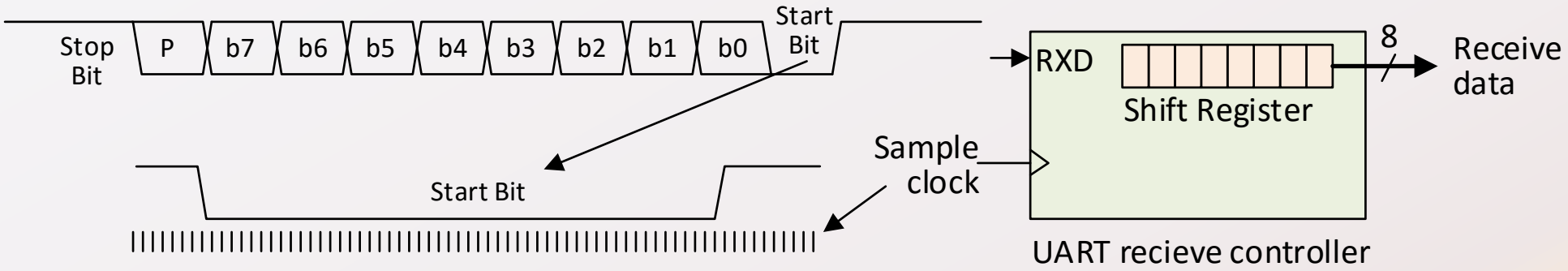
Asynchronous Communications

Receiver uses faster sample clock to record the TXD line, so it gets many samples per bit

When 1-0 transition detected, waits ½ bit-clock, then samples again. If still low, then start bit

Takes new sample every bit-clock time, in the middle of each bit

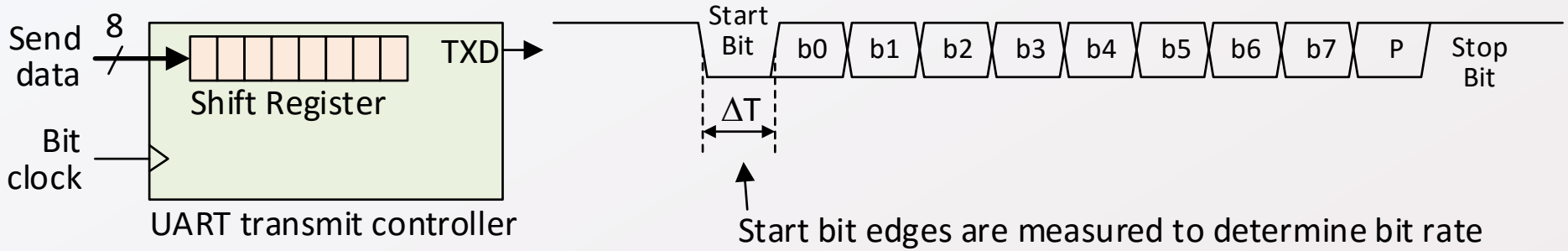


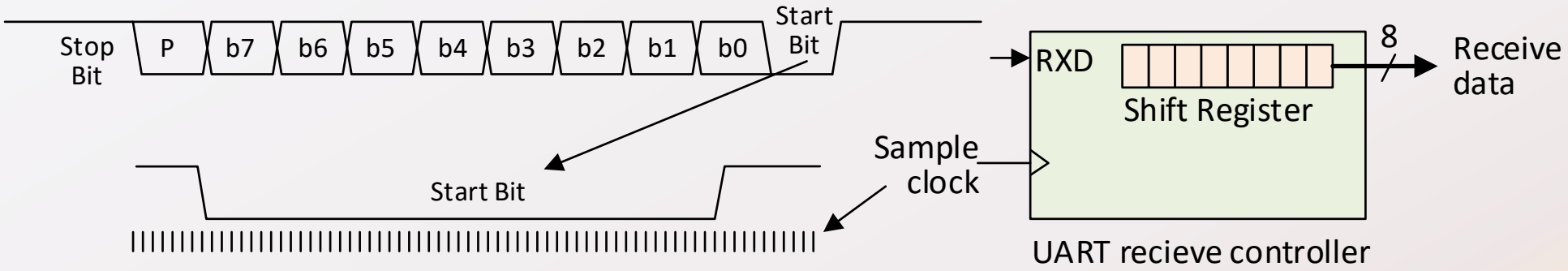


Asynchronous Communications

Serial data assembled into parallel byte for reading by processor

“Bit times” measured as possible, and receiver sample time modified on fly as needed





UART Communications

"Baud rate" means symbols per second; for binary symbols, equivalent to bits/second

Based on early PC clock frequencies, several "standard" baud rates have emerged:
300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000, 256000

Newest systems can go up to 5MHz

Most UART systems are human-speed, so don't need high data rates

UARTs are simple, widely available, drivers exist… why change?

Common applications and data rates…

ZYNQ/ARM UART System

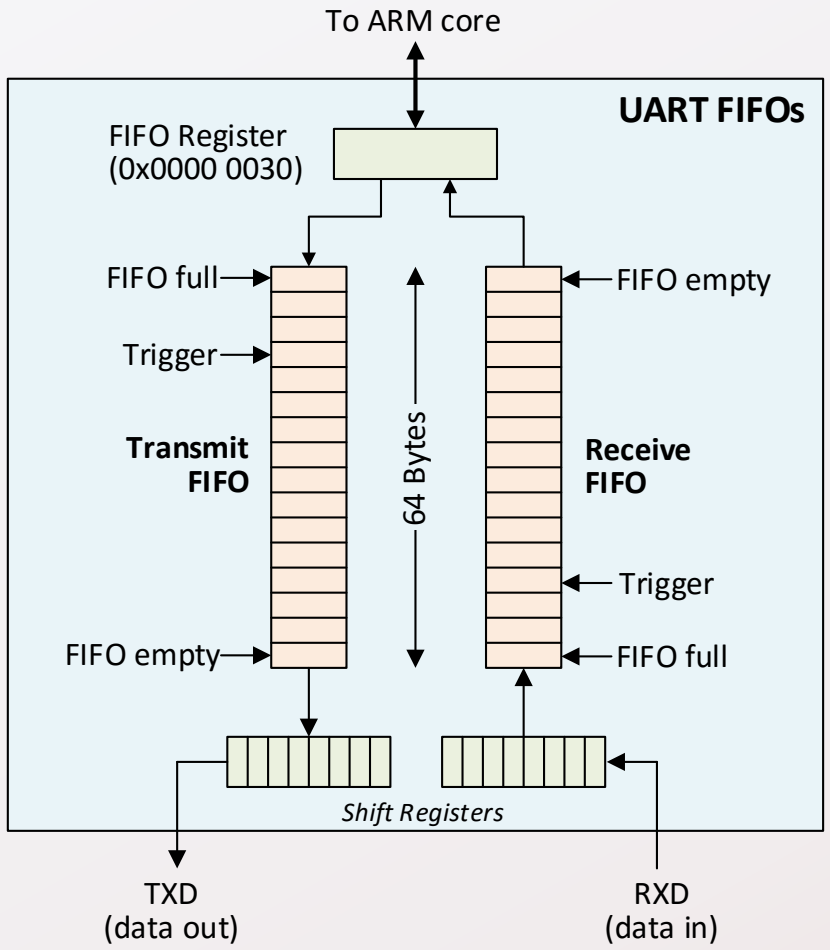
Two UART controllers that are both memory-mapped peripherals

The main registers (see ZYNQ TRM or BPR for more info)

Name	Function	Address	Bits
CR (Control Regsiter)	Control (enables, resets, breaks)	0x 0000 0000	8
MR (Mode Register)	Mode (parity, # stop bits, # char bits, clk source)	0x 0000 0004	10
IER (Interrupt Enable Reg.)	Enables possible interrupt sources	0x 0000 0008	13
IDR (Interrupt Disable Reg.)	Disables possible interrupt sources	0x 0000 000C	13
IMR (Interrupt Mask Reg.)	Mask bits for possible interrupt sources	0x 0000 0010	13
ISR (Interrupt Status Reg.)	Indicates which interrupt are currently asserted	0x 0000 0014	13
BAUDGEN (Clock divider)	Baud rate clock divider for main sample clock	0x 0000 0018	16
RXTOUT (Idle time set)	Sets receiver idle time before possible interrupt issued	0x 0000 001C	8
RXWM (Receive fill level)	Sets receive FIFO fill level before possible interrupt issued	0x 0000 0020	6
SR (FIFO status)	FIFO status register (full, empty, near full, etc.)	0x 0000 002C	6
FIFO (Read/write FIFO data)	Transmit and receive FIFO data port	0x 0000 0030	8
Baud_Rate_D (Clock divider)	Divides BAUDGEN clock to define bit-clock	0x 0000 0034	8
TX_Trigger (Transmit fill level)	Sets transmibt FIFO fill level before possible interrupt issued	0x 0000 0044	6

ZYNQ's UART FIFOs

TXD and RXD path have 64-byte FIFOs



ZYNQ's UART FIFOs

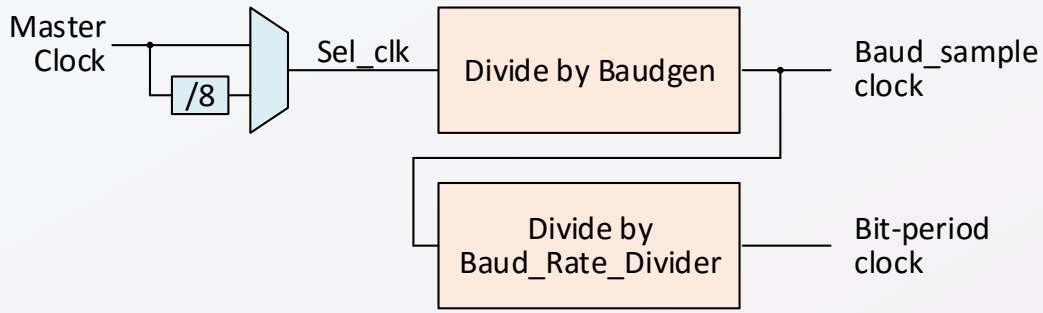
Status Bits in FIFO Status Register		
Name	**Function**	**Bit**
TNFUL	Transmitter nearly full	14
TTRIG	Transmitter trigger state ('0' less than TTRIG, '1' greater)	13
FLOWDEL	Receiver flow delay ('0' less than FDEL, '1' greater)	12
TACTIVE	Transmitter active	11
RACTIVE	Receiver active	10
TXFULL	Transmitter full	4
TXEMPTY	Transmitter empty	3
RXFULL	Receiver full	2
RXEMPTY	Receiver empty	1
RXOVR	Receiver trigger state ('0' less than RTRIG, '1' greater)	0

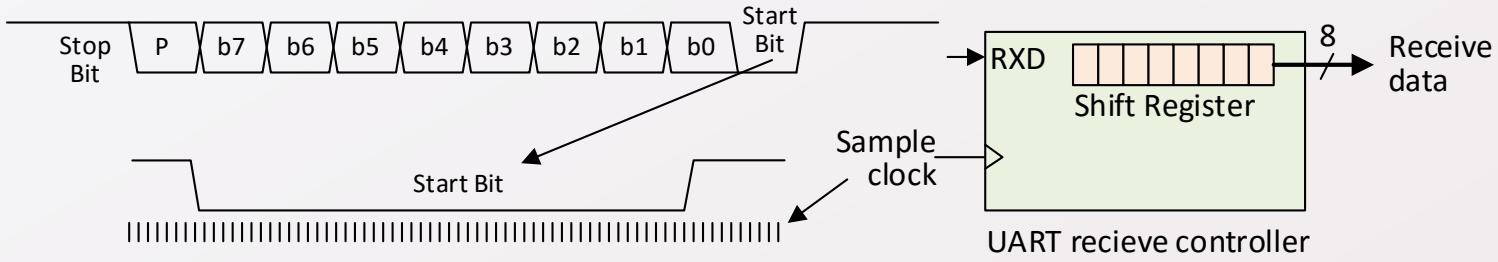
ZYNQ/ARM UART System

Several relevant interrupts available

Interrupt Functions		
Name	**Function**	**Bit**
TOVR	Transmitter overflow	12
TNFUL	Transmitter FIFO nearly full	11
TTRIG	Transmitter FIFO trigger	10
DMS	Modem status	9
TOUT	Receiver timeout	8
PARITY	Parity error	7
FRAMING	Framing error	6
OVERFLOW	Receiver overflow	5
TFULL	Transmitter full	4
TXEMPT	Transmitter empty	3
RXFULL	Receiver FIFO full	2
RXEMPT	Receiver FIFO empty	1
RXOVER	Receiver FIFO trigger	0

Baud Generators





COM ports

First used/named by IBM PC in the early 1980s

Refers to ports and signals plus driver; UART assumed

Developed for modems, but then widely used for other things

No physical transport defined

Terminal Programs (COM driver interface)

Configuring ZYNQ's UART

1. Reset the UART Controller (Control Register)
2. Set the BAUDGEN clock divider (115200 baud works well; 0x7C divider value)
3. Set the Buad_Rate_Divider for bit-clock (again for 115200, 0x6 works)
4. Enable software resets, transmitter & receiver, Stop Transmitter Break (Control Register)
5. Channel mode to normal, 1 stop bit, no parity, character length 8 bit, clock source to UART reference clock (Mode register)
6. Set timeout to 1 (RXTOUT register)
7. Disable all interrupts (IDR register)