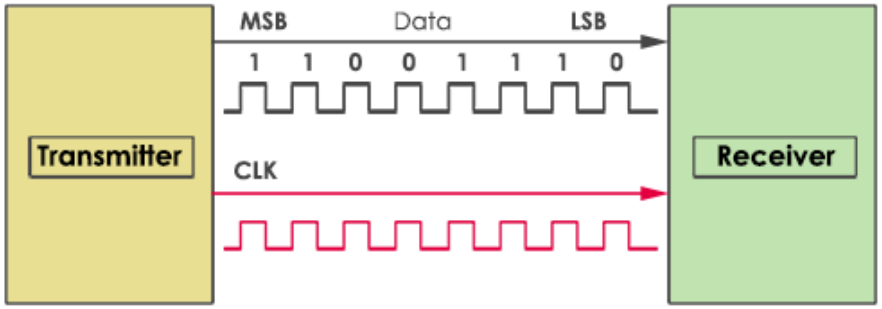
Universal Asynchronous Receiver-Transmitter (UART)

Serial Interfaces & Protocols

Serial Protocol

A **Serial Protocol** defines the process of Sending or Receiving data one bit at a time sequentially over a communications channel.



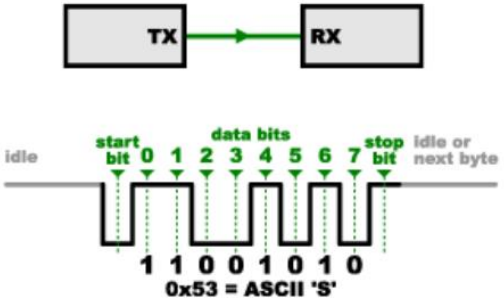
Why Use Serial?

- Why do we use Serial transmission protocols?
  - Ideal over long distances & lower cost than parallel transmission - less bulky, less wires.
  - Higher data throughput with long distances:
    - Less line capacitances => more bits per unit time.

- However, why NOT Serial?
  - More complex interfacing logic (electronics circuitry) & communication protocol (software/ firmware).
    - Transmitter needs to de-compose bytes to bits.
    - Receiver needs to re-compose bits back to bytes.
    - Control bits (e.g. for flow control, acknowledgement) need to be sent on the same wire.

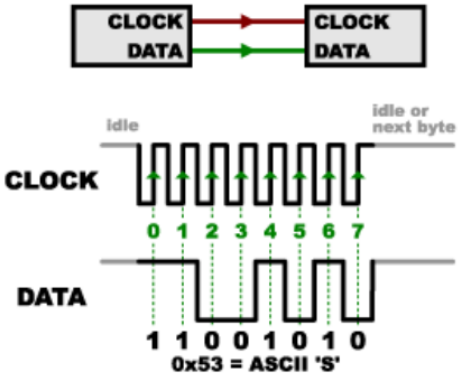
Types of Serial Communications

- Two Basic Serial Types:
  - **Asynchronous:** (e.g UART)
    - Regulated by control signals along the transmission lines.
    - Preset frequency between devices (baud rate) – *transmitter & receiver needs to know the frequency to use.*
    - Frequency set at beginning of data transfer.



asynchronous

  - **Synchronous:** (e.g SPI, I^2C)
    - Regulated by an external **clock**.
    - Requires an extra wire for clock signal.
    - Most implementations utilizes TWO different signals on separate lines:
      - CLOCK & DATA signals

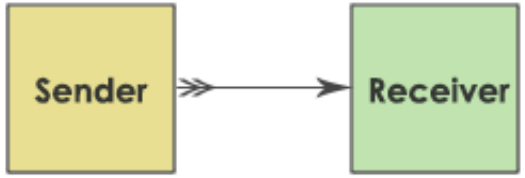


synchronous

Serial Transmission Types

Simplex method:

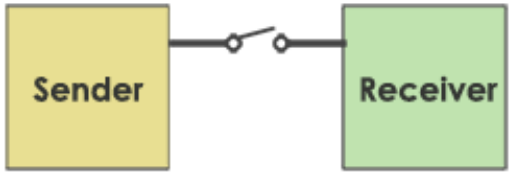
- One-way communication.
- Only one, Sender or Receiver, can be active at a time.
- If a sender transmits, the receiver can only accept.
- Example: Radio & TV transmission.



Simplex

Half Duplex mode:

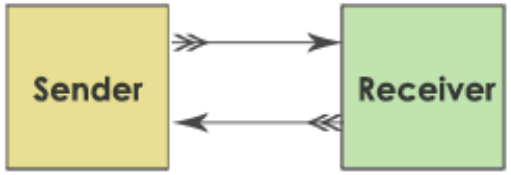
- Both Sender & Receiver can be active but not at same time.
- If a sender transmits, the receiver can accept but cannot send & vice versa.
- Example: Internet surfing.



Half Duplex

Full Duplex mode:

- Both Sender & Receiver can transmit & receive at the same time.
- Example: telephone networks (smartphone).



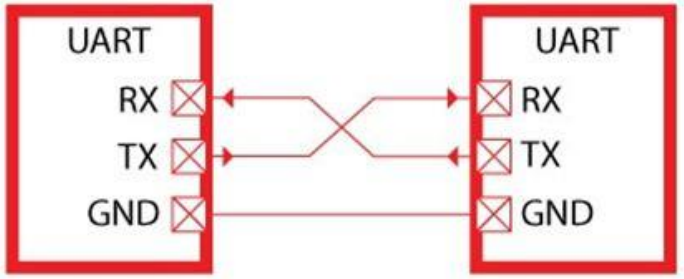
Full Duplex

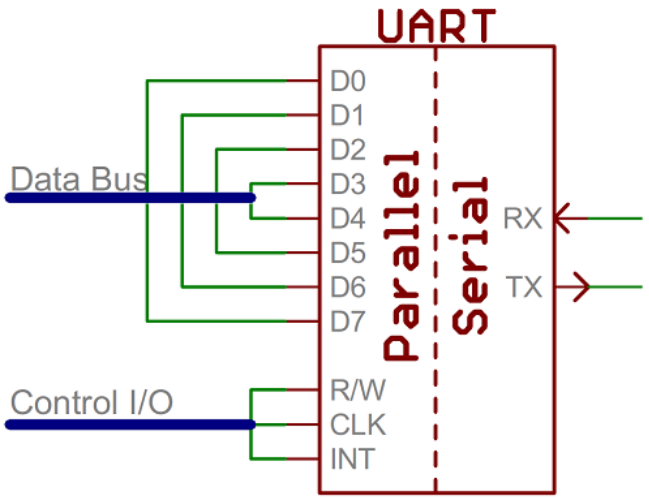
UART

Universal **A**synchronous **R**eceiver-**T**ransmitter

UART

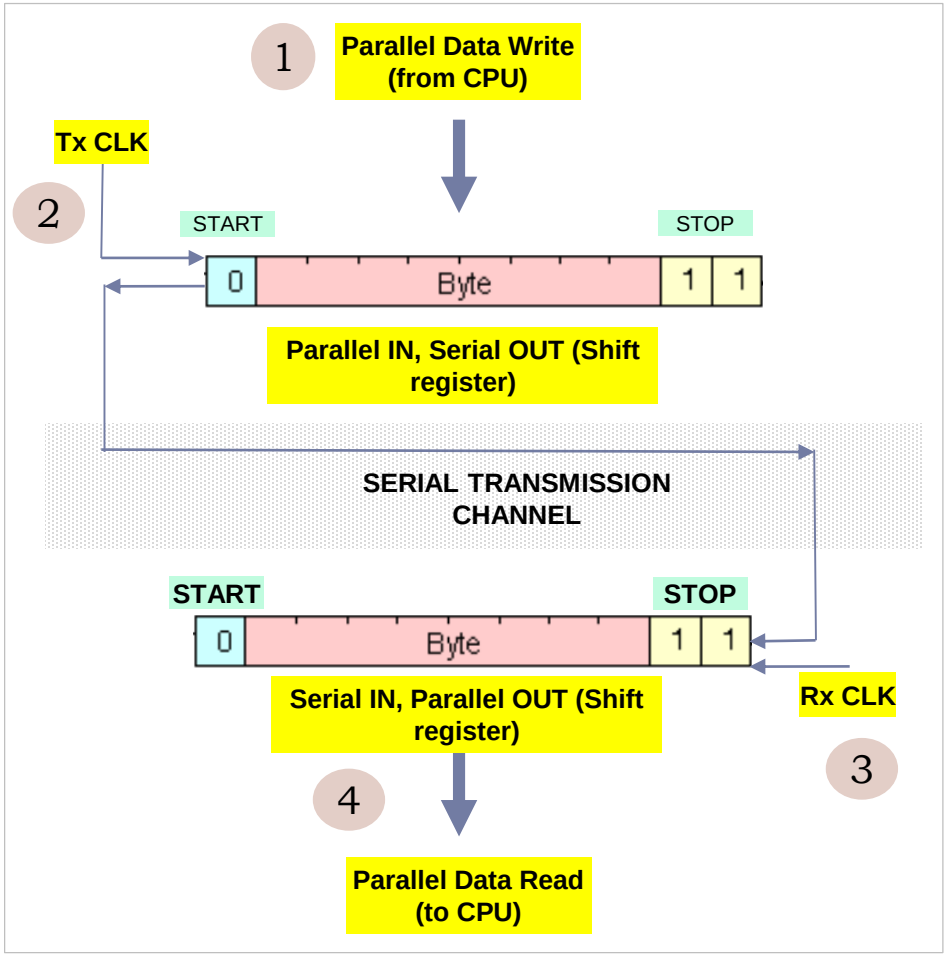
- **UART**: **U**niversal **A**synchronous **R**eceiver/ **T**ransmitter.
- Basic function of a UART: Converts parallel data to serial data *(transmit)* & vice-versa *(receive)*.
- Implemented using shift registers.
- Shift registers converts between serial & parallel data formats.





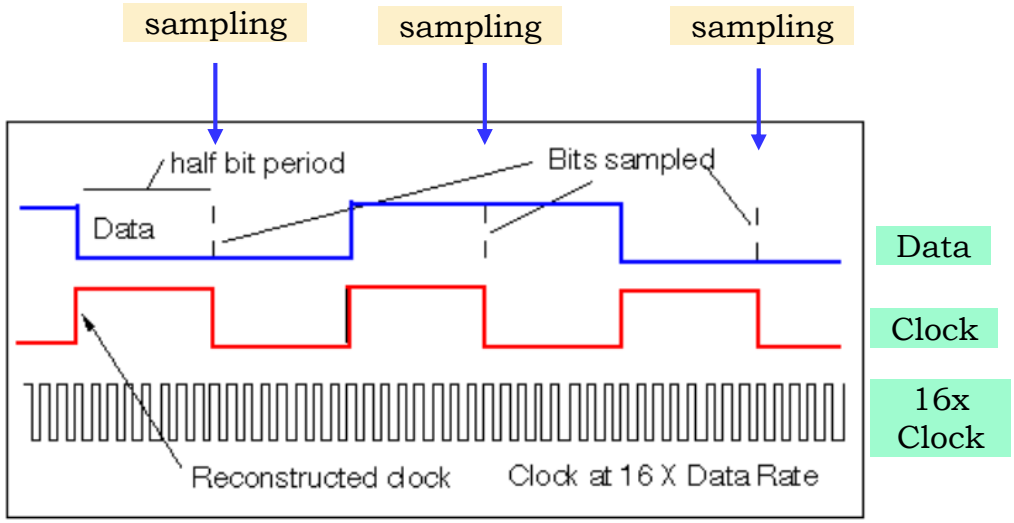
UART Data Transmission

1. Data to be transmitted is loaded in parallel, usually from a transmit FIFO.
2. Data is sent by supplying a Transmit Clock (Tx Clock) to shift the data out serially through a **Shift register**.
    1. START bit is shifted out first, followed by the LSB bit of the data bit.
3. At the UART Receiver, a Receive Clock (Rx CLK) is used to clock-in the data to the shift register.
    1. Rx CLK must be of the same ***nominal*** frequency as the Transmit clock (Tx CLK).
4. Data is read in parallel format by the CPU.



UART Data Receive

- Receiver detects the START bit (logic 'L') of the incoming data. Once detected, data is **sampled by a higher frequency clock signal**, usually **8X** or **16X** the Rx CLK.
- When all the data bit (START, data, parity, STOP) have been shifted into the shift register, it can be read in by the CPU.



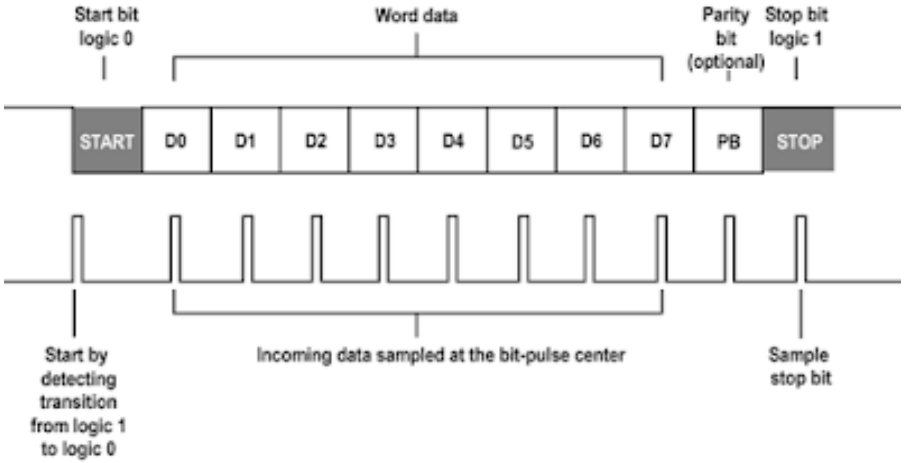
A note about Tx CLK & Rx CLK in asynchronous transmission:

The clock speeds must be **pre-determined**, i.e., the Receiver needs to know the Transmitter speed used.

Both Transmit & Receive clock speeds **need not be exact** *(therefore, 'same nomimal speed' is used in previous slide)* .

Asynchronous data frames are short (less number of bits) & clock drift is not a issue.

UART Data Frame



- **START** bit:
  - Alerts receiver of incoming data.
  - Synchronizes clock for data transfer.
- **STOP** bits:
  - Can be 1 or 2 bits.
- **PARITY** bits: Even, Odd, Stick or No Parity.
  - Used for error checking.
- **DATA** bits: can be 5, 6, 7 or 8 bits in length.
  - Not all data requires 8-bits. E.g. ASCII codes are in 7 bits.

Error Detection and Correction

- **Error detection** is the ability of the receiver to detect errors during transmission.
- **Error correction** is the ability of receiver and transmitter to cooperate to correct problem.
- Correction typically done by acknowledgement/re-transmission protocol.
- Bit error: single bit is inverted
- Burst of bit error: consecutive bits received incorrectly
- **Parity**: extra bit sent with word used for error detection
  - Odd parity: data word plus parity bit has odd number of 1's. (data = 1011.1100 => parity bit = 0).
  - Even parity: data word plus parity bit has even number of 1's. (data = 1011.1100 => parity bit = 1).
  - Always detects single bit errors; not able to detect burst bit errors.
  - Stick parity refers to a permanent '1' or '0' being sent as parity. Not useful for error detection as it does not reflects error status.

UART Implementation in TM4C123G

8 UARTS: UART0 to UART7
UARTDR, UARTFR registers

UART Implementation (TM4C123G)

- Total of **8 possible UARTs** on TM4C123G.
  - **UART0 to UART7**.
  - UART0 pins are not available on the LaunchPad board connectors. It is implemented as a Virtual COM port through the USB programming port.
- Supports serial speeds up to 5Mbps or 10Mbps (depending on clock division (*8* / *16*) used).
- Performs parallel-to-serial & serial-to-parallel conversions.
- Error Checking through parity bits.
- Separate Transmit and Receive FIFOs (16 levels, for data buffering).
- **Internal UART clock** for data Transmit/Receive.
  - Clock in UART runs faster than data rate.
  - Usually **8** × or **16** × of data clock rate (programmable).

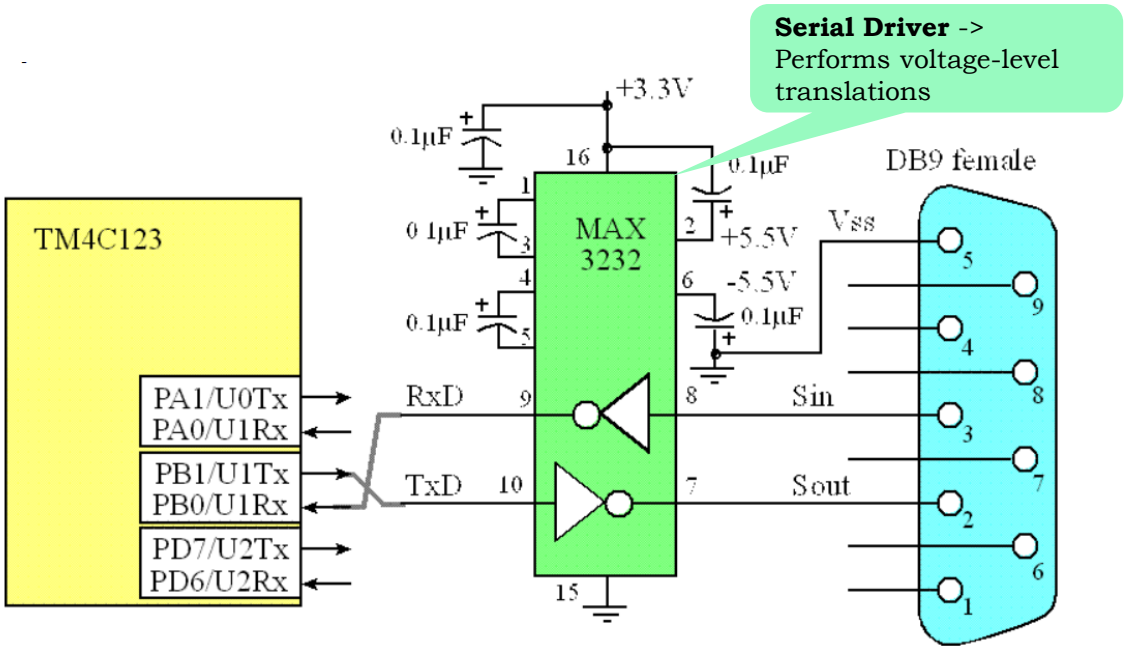
UART Implementation (TM4C123G)

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
U0Rx	17	PA0 (1)	I	TTL	UART module 0 receive.
U0Tx	18	PA1 (1)	O	TTL	UART module 0 transmit.
U1CTS	15 29	PC5 (8) PF1 (1)	I	TTL	UART module 1 Clear To Send modem flow control input signal.
U1RTS	16 28	PC4 (8) PF0 (1)	O	TTL	UART module 1 Request to Send modem flow control output line.
U1Rx	16 45	PC4 (2) PB0 (1)	I	TTL	UART module 1 receive.
U1Tx	15 46	PC5 (2) PB1 (1)	O	TTL	UART module 1 transmit.
U2Rx	53	PD6 (1)	I	TTL	UART module 2 receive.
U2Tx	10	PD7 (1)	O	TTL	UART module 2 transmit.
U3Rx	14	PC6 (1)	I	TTL	UART module 3 receive.
U3Tx	13	PC7 (1)	O	TTL	UART module 3 transmit.
U4Rx	16	PC4 (1)	I	TTL	UART module 4 receive.
U4Tx	15	PC5 (1)	O	TTL	UART module 4 transmit.
U5Rx	59	PE4 (1)	I	TTL	UART module 5 receive.
U5Tx	60	PE5 (1)	O	TTL	UART module 5 transmit.
U6Rx	43	PD4 (1)	I	TTL	UART module 6 receive.
U6Tx	44	PD5 (1)	O	TTL	UART module 6 transmit.
U7Rx	9	PE0 (1)	I	TTL	UART module 7 receive.
U7Tx	8	PE1 (1)	O	TTL	UART module 7 transmit.

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 895

UART Implementation (TM4C123G)



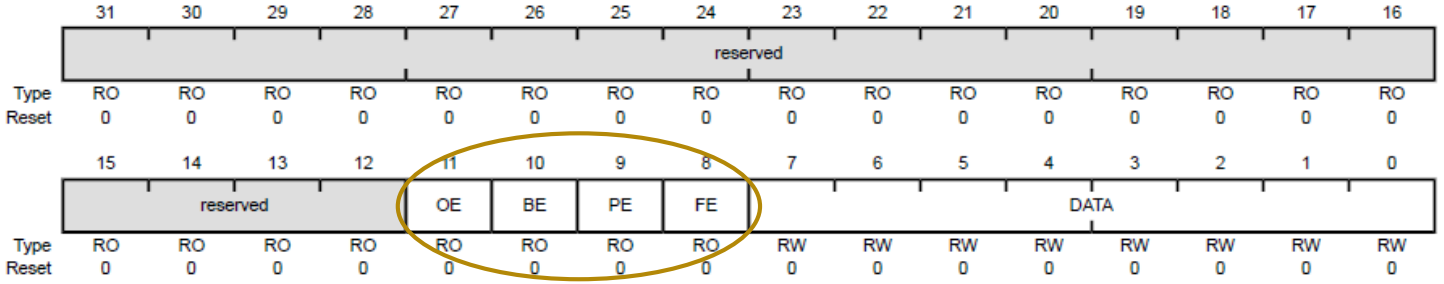
- Diagram shows UARTS U0, U1 and U2.
  - **UART0**: Transmit (PA1), Receive (PA0).
  - **UART1**: Transmit (PB1 or PC5), Receive (PB0 or PC4).
  - **UART2**: Transmit (PD7), Receive (PD6).
- Voltage level shifter is usually needed between TTL logic and serial (UART) port.

Note: UART1 can be mapped in TWO ways!

UART Data Register (UARTDR)

UART Data (UARTDR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x000
Type RW, reset 0x0000.0000



Error Status bits

- Data written to register is pushed to the Transmit FIFO (if enabled).
- During a Read, 8-bits of Data & 4-bits of Status are pushed to the Receive FIFO. Total of 12-bits.
- 4 Status bits: Break Error (**BE**), Framing Error (**FE**), Parity Error (**PE**), Overrun Error (**OE**).

Write-data is 8-bits wide.

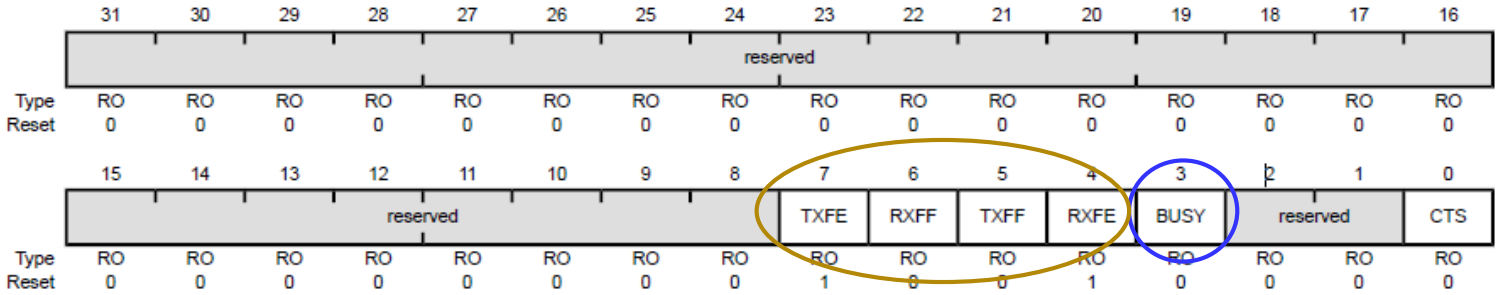
Read-data is 12 bits wide.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 906

UART Flag Register (UARTFR)

UART Flag (UARTFR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x018
Type RO, reset 0x0000.0090



FIFO Status bits:

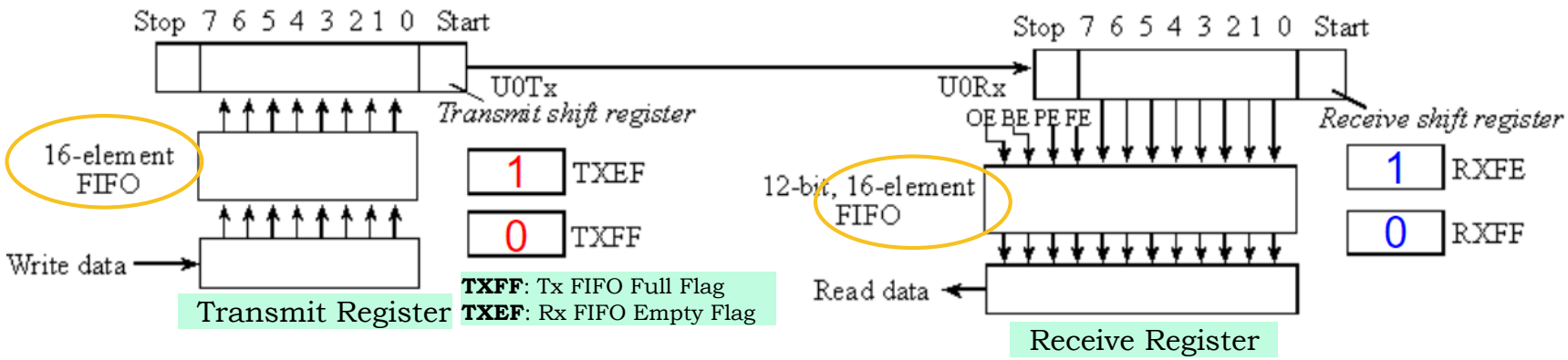
- Bit 4: **RXFE** – Receive FIFO Empty
- Bit 5: **TXFF** – Transmit FIFO Full
- Bit 6: **RXFF** – Receive FIFO Full
- Bit 7: **TXFE** – Transmit FIFO Empty

BUSY bit:

- Bit 0: UART is not Busy
- Bit 1: UART is Busy transmitting Data. **Bit remains set until the data byte is fully transmitted (including Stop bits) from shift register**.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 906

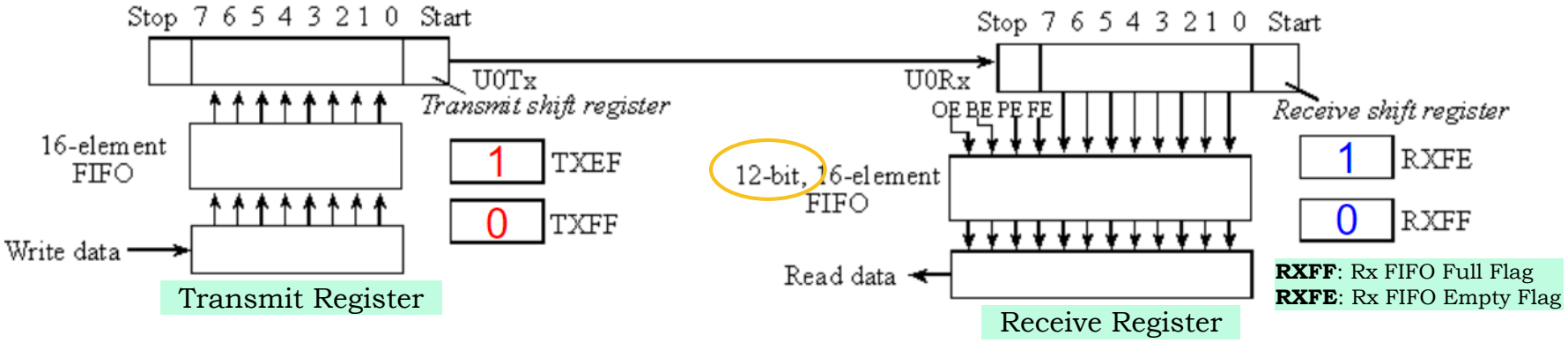
UART Data Transmission (TM4C123G)



- SW checks that the transmit FIFO is not full (**TXFF=0** in the Flag register (**UARTFR**)) & then writes to Transmit Data register (**UARTDR**).
- Data bits are then shifted out in following order:
  START, $\mathbf{b_0}$, $\mathbf{b_1}$, $\mathbf{b_2}$, $\mathbf{b_3}$, $\mathbf{b_4}$, $\mathbf{b_5}$, $\mathbf{b_6}$, $\mathbf{b_7}$, **STOP**, where $\mathbf{b_0}$ is LSB, $\mathbf{b_7}$ is MSB.
- Transmit Data is a Write-Only register.
- SW can write to data register (**UARTDR**) if its **TXFF** (Transmit FIFO full) flag is 0.
  - **TXFF**=0 means FIFO is not full & has room for more data.

Tx & Rx FIFOs are 16-levels deep.

UART Data Receive (TM4C123G)

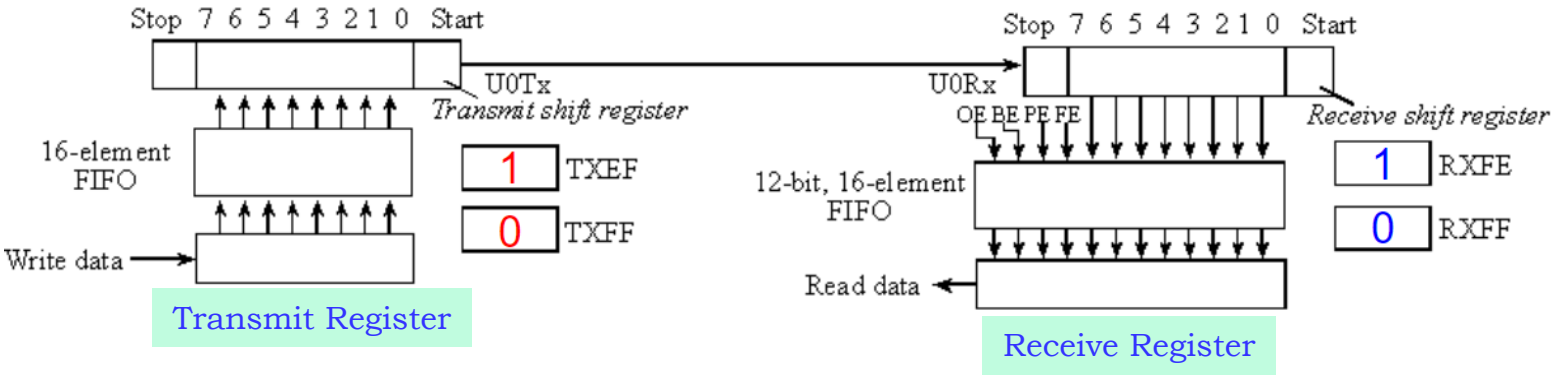


- Receive Data register (**UARTDR**) has the same address as the Transmit Data register.
- However, TX & RX Data registers are implemented as 2 separate registers.

Tx FIFO is 8-bits wide.
Rx FIFO is 12-bits wide.

- SW reads incoming data from the **UARTDR** register if the **RXFE** (receive FIFO empty) flag in the Flag Register (**UARTFR**) register is 0. It means the FIFO is not empty & has unread data.

UART TX & RX FIFO (TM4C123G)



- **Transmit FIFO**: **16 X 8 FIFO**.
  - Transmit data is stored in FIFO through writing to the **UARTDR** register. UART HW will send data serially in the right order.
- **Receive FIFO**: **16 X 12 FIFO**.
  - Reading the UARTDR register returns **12 bits** from the FIFO (8 bits of data & 4 error flags).
  - Receive shift register & receive FIFO are separate from those in the transmitter.
  - UART receiver recognizes a new frame by the START bit.
  - Data bits are shifted in through same order as the Transmitter shifted them out: **START**, $\mathbf{b_0}$, $\mathbf{b_1}$, $\mathbf{b_2}$, $\mathbf{b_3}$,$\mathbf{b_4}$, $\mathbf{b_5}$, $\mathbf{b_6}$, $\mathbf{b_7}$, **STOP**.
- FIFOs are managed by the HW. It cannot be accessed directly by SW.

UART Baud Rate Generation

UARTCTL, UARTIBRD, UARTFBRD, UARTLCRH registers

UART Baud Rate Generation (TM4C123G)

- Baud Rate is generated from the System Clock together with a Clock Divisor (*ClkDiv*).
- Baud Rate Divider (BRD) is a **22-bit divisor** made up of an Integer part & Fractional part.
  - Integer part = **DIVINT** = 16-bits field (**UARTIBRD** register).
  - Fractional part = **DIVFRAC** = 6-bits field (**UARTFBRD** register).

$$Baud\ Rate\ Divider\ (BRD) = \frac{System\ Clock}{ClkDiv\ \times Baud\ Rate}$$

- Clock Divisor *(ClkDiv)* is either 8 or 16.
  - Selected through **HSE** bit in UART Control register (**UARTCTL**).
  - If **HSE** = '1', divisor value is 8.
  - If **HSE** = '0', divisor value is 16.

[Refer to TM4C123G datasheet, pg 896 for more details]

UART Control Register (UARTCTL)

UART Control (UARTCTL)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x030
Type RW, reset 0x0000.0300

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CTSEN	RTSEN	reserved		RTS	reserved	RXE	TXE	LBE	reserved	HSE	EOT	SMART	SIRLP	SIREN	UARTEN
Type	RW	RW	RO	RO	RW	RO	RW	RW	RW	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

HSE: Controls Baud Rate Clock Division.
'1' = Clock/8;
'0' = Clock/16

- Upon Reset, all bits are cleared except for Transmit Enable (TXE) & Receive Enable (RXE) bits, which are set.
- **UARTEN**: '1': enable UART module
- Note: If configuration change in the module is needed, **UARTEN** bit must be cleared before configuration changes are written.
- Note: If a UART is disabled during a transmit or receive operation, the current transaction is completed prior to UART stopping.

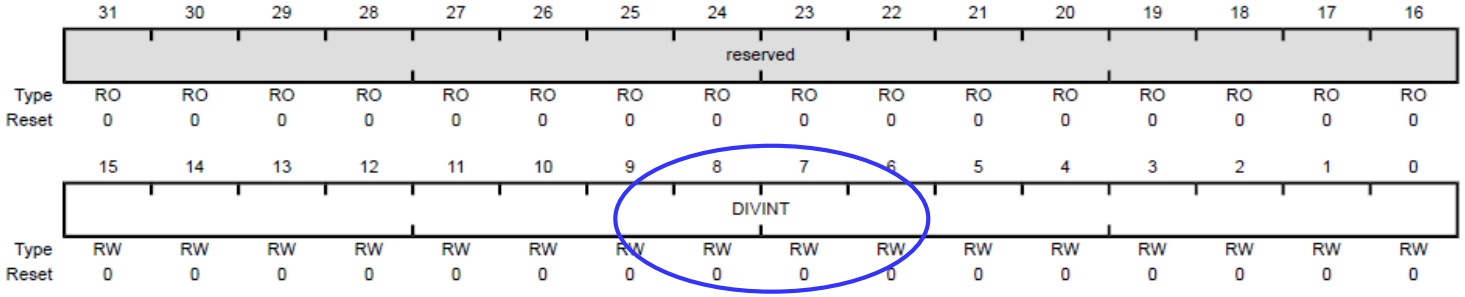
Bit
0: **UARTEN** – UART Enable
4: **EOT** – End of Transmission
5: **HSE** – High Speed Enable
8: **TXE** – UART Transmit Enable
9: **RXE** – UART Receive Enable

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 918

UART Integer Baud-Rate Divisor Register (UARTIBRD)

UART Integer Baud-Rate Divisor (UARTIBRD)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x024
Type RW, reset 0x0000.0000



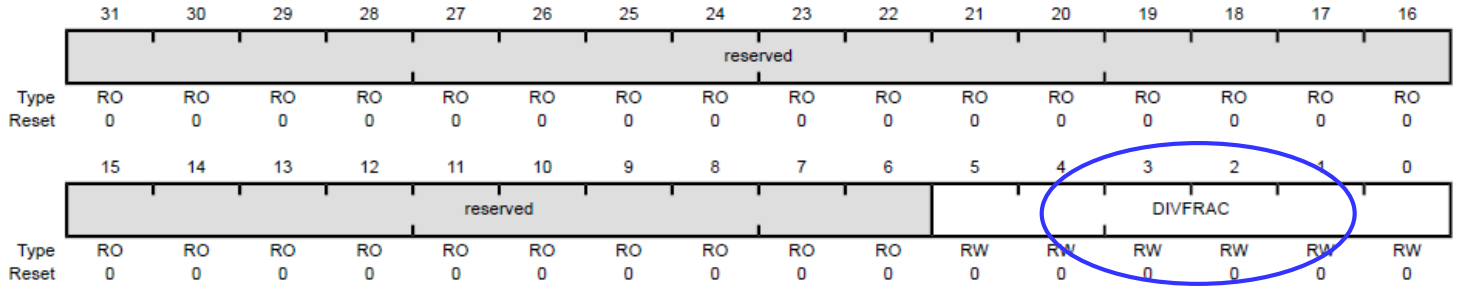
- **Integer** part of Baud Rate Divider = **DIVINT** = 16 bits.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 914

UART Fractional Baud-Rate Divisor Register (UARTFBRD)

UART Fractional Baud-Rate Divisor (UARTFBRD)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x028
Type RW, reset 0x0000.0000



- **Fractional** part of Baud Rate Divider = **DIVFRAC** = 6 bits.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 915

Ex 1: UART Baud Rate Generation

Note: In Tiva LaunchPad, Baud rate and Bit rate is used interchangeably.

Example: If system clock = 80 MHz. How can we obtain a UART baud rate of 19,200 bits/s.

Assuming ***ClkDiv*** = 16, we can calculate the baud rate divider as below,

$$Baud\ Rate\ Divider\ (BRD) = \frac{80{,}000{,}000}{16 \times 19{,}200} = 260.4167$$

Integer part = **DIVINT** = 260

Fractional part = **DIVFRAC** = int (0.4167 x 64 + 0.5)

= 27

Equivalent to <<6: convert decimal field to a 6-bit number

To bring to next integer value

The 6-bit fractional number (that is to be loaded into the `DIVFRAC` bit field in the **UARTFBRD** register) can be calculated by taking the fractional part of the baud-rate divisor, multiplying it by 64, and adding 0.5 to account for rounding errors:

```
UARTFBRD[DIVFRAC] = integer(BRDF * 64 + 0.5)
```

[from TM4C123G datasheet, pg 896]

Ex 1: UART Baud Rate Generation

Example *(continue)*: What is the actual baud we would get with the calculated values?

$$Actual\ Baud\ Rate = \frac{80{,}000{,}000}{16\ \times (260 + \frac{27}{64})} = 19{,}199.61\ bits/s$$

$$Error\ Rate = \frac{(19{,}200 - 19{,}199.61)}{19{,}200} \times 100\% = 0.002\%$$

As a guide, Baud Rates implemented should be within ±5% of intended values for reliable communications.

[Refer to TM4C123G datasheet, pg 896 for more details]

Ex 2: UART Baud Rate Generation

Example: If system clock = 10 MHz, what Baud Rate is obtain if we load register fields DIVINT = 2 & DIVFRAC = 32?

Assume clock division of 16.

Integer part = **DIVINT** = 2

Fractional part = **DIVFRAC** = 32

$$Baud\ Rate = \frac{10{,}000{,}000}{16\ \times (2 + \frac{32}{64})} = 250\ Kbits/s$$

Ex 3: UART Baud Rate Generation

Example: If system clock = 50 MHz, determine the register field values to obtain a Baud Rate of 38,400 bits/s. Assume clock division of 16.

What is the actual baud rate implemented and error rate?

$$Baud\ Rate\ Divider\ (BRD) = \frac{50{,}000{,}000}{16\ \times 38{,}400} = 81.3802$$

Integer part = **DIVINT** = 81

Fractional part = **DIVFRAC** = int (0.3802 × 64 + 0.5) = 24

$$Actual\ Baud\ Rate = \frac{50{,}000{,}000}{16\ \times (81 + \frac{24}{64})} = 38{,}402.458\ bits/s$$

$$Error\ Rate = \frac{\langle 38{,}402.458 - 38{,}400 \rangle}{38{,}400} \times 100\% = 0.0064\%$$

UART Line Control Register (UARTLCRH)

- UART transmit/receive parameters are set through the Line Control register (**UARTLCRH**).
- Parameters includes,
  - Number of Stop bits,
  - Parity,
  - Whether FIFO is used, …..

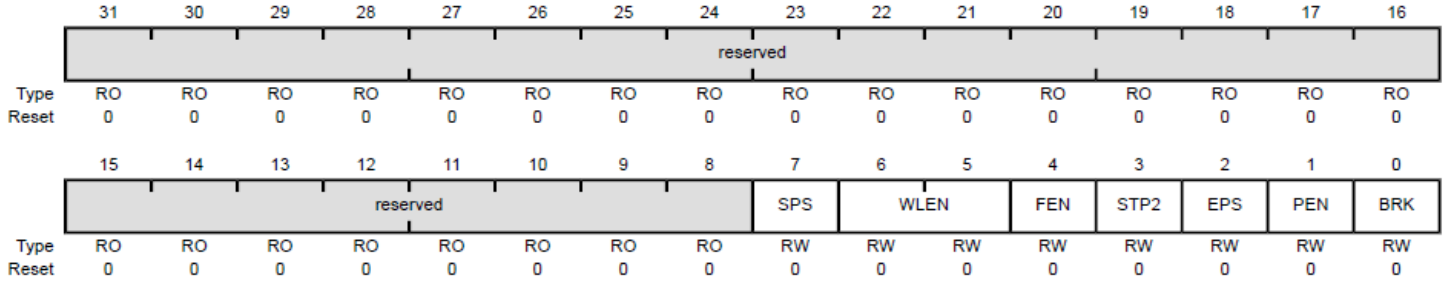
Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 915

UART Line Control Register (UARTLCRH)

UART Line Control (UARTLCRH)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x02C
Type RW, reset 0x0000.0000

Bit
1: **PEN** – Parity Enable
2: **EPS** – Even Parity Select
3: **STP2** – Use 2 STOP Bits
4: **FEN** – FIFO Enable
6:5: **WLEN** – Word Length (No of bits/data)



- **UARTLCRH** register is the line control register to set serial parameters such as **data length** (5,6,7,8 bits), **parity** & **stop** bits selection & **FIFO** control.
  - **WLEN**: 0x00 -5 data bits (default), 0x01 -6 bits, 0x02 -7 bits, 0x03 -8 bits.
  - **STP2**: '0' – 1 Stop bit, '1' – 2 Stop bits.
  - **EPS**: '0' –Odd parity, '1' –Even parity.
  - **FEN**: '1' – enable FIFO
- Note: When updating the baud-rate divisor (**UARTIBRD** and/or **UARTFBRD** registers), the **UARTLCRH** register must also be written for baud rate to take effect

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 915

UART Initialization

RCGCUART, PRUART, UARTCTL, UARTIBRD, UARTFBRD, UARTLCRH, UARTCC

UART Initialization

Two broad steps:

- Enable & initialize **UART GPIO** pins.
- Initialize **UART** module.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 902

UART Initialization

Steps to enable & initialize **UART GPIO** pins:

1. Enable clock to the appropriate GPIO module through **RCGCGPIO & PRGPIO** register *(pg 340 of datasheet)*.

2. Set GPIO for Alternate function: **AFSEL** bits for the appropriate pins *(pg 671)*.

3. Configure GPIO pin drive level and/or slew rate for the mode selected (*if needed*).

4. Map the Alternate function through the **PMC*n*** fields in the **GPIOPCTL** register to assign the UART signals to the appropriate GPIO pins *(pg 688, pg 1351)*.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 902

UART Initialization

Steps to enable & initialize **UART module** (*continued*):

1. To use the UART, the peripheral clock must be enabled by setting the appropriate bit in the **RCGCUART** register (*pg 344*) & check that UART is ready for access through the **PRUART** register *(pg 410)*.
2. Determine the **BRD** values for the required Baud Rate.
3. Disable the UART by clearing the **UARTEN** bit in the **UARTCTL** register.
4. Write the integer portion of the **BRD** to **UARTIBRD** register.
5. Write the fractional portion of the **BRD** to **UARTFBRD** register.
6. Write the desired serial parameters to the **UARTLCRH** register (data length, parity & stop bit, FIFO use).
7. Configure UART clock source by writing to the **UARTCC** register.
8. Enable UART by setting the **UARTEN** bit in the **UARTCTL** register.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 903

UART Clock Gating Register (RCGCUART)

Universal Asynchronous Receiver/Transmitter Run Mode Clock Gating Control (RCGCUART)

Base 0x400F.E000
Offset 0x618
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								R7	R6	R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Enables & Disables clock to the UART modules. Clock is needed to access the UART registers.
- **R7** to **R0** denotes UART 7 to 0.
- '0' – Clock to UART is disabled.
- '1' – Clock to UART is enabled.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 344

UART Peripheral Ready Register (PRUART)

Universal Asynchronous Receiver/Transmitter Peripheral Ready (PRUART)

Base 0x400F.E000
Offset 0xA18
Type RO, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								R7	R6	R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Indicates whether UART is ready for access.
- **R7** to **R0** denotes UART 7 to 0.
- '0' – UART is not ready for access.
- '1' – UART is ready for access.

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 410

UART Control Register (UARTCTL) *(see also slide 26)*

UART Control (UARTCTL)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x030
Type RW, reset 0x0000.0300

Bit
0: **UARTEN** – UART Enable
4: **EOT** – End of Transmission
5: **HSE** – High Speed Enable
8: **TXE** – UART Transmit Enable
9: **RXE** – UART Receive Enable

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CTSEN	RTSEN	reserved		RTS	reserved	RXE	TXE	LBE	reserved	HSE	EOT	SMART	SIRLP	SIREN	UARTEN
Type	RW	RW	RO	RO	RW	RO	RW	RW	RW	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

- **UARTEN**: '1': UART enabled; '0' UART disabled.
- **EOT**: '0': TXRIS bit set when FIFO condition triggered; '1' TXRIS bit set after all bits sent.
- **HSE**: Sets UART clock. '0': System clock/16; '1' System clock/8.
- **TXE**: '1': UART transmitter enabled; '0' UART transmitter disabled.
- **RXE**: '1': UART receiver enabled; '0' UART receiver disabled.

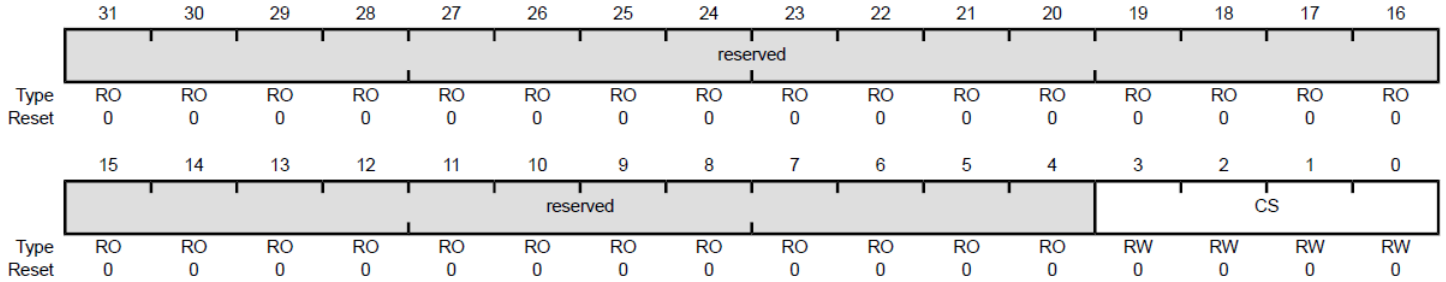
Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 918

UART Clock Configuration (UARTCC)

UART Clock Configuration (UARTCC)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0xFC8
Type RW, reset 0x0000.0000

UARTCC selects UART Clock Source through the CS bits:
- System Clock (default) or PIOSC



CS	RW	0

UART Baud Clock Source

The following table specifies the source that generates for the UART baud clock:

Value	Description
0x0	System clock (based on clock source and divisor factor)
0x1-0x4	reserved
0x5	PIOSC
0x5-0xF	Reserved

Source: TM4C123GH6PM Datasheet (spsm376e.pdf) pg 939

UART Data Structure

[file: TM4C123GH6PM7.h]

0x4000.C000	DR
0x4000.C004	ECR_UART_ALT
0x4000.C008	RSR
0x4000.C018	FR
0x4000.C020	ILPR
0x4000.C024	IBRD
0x4000.C028	FBRD
0x4000.C02C	LCRH
0x4000.C030	CTL
0x4000.C034	IFLS
0x4000.C038	IM
0x4000.C03C	RIS
0x4000.C040	MIS
0x4000.C044	ICR
0x4000.C048	DMACTL
0x4000.C0A4	_9BITADDR
0x4000.C0A8	_9BITAMASK
0x4000.CFC0	PP
0x4000.CFC8	CC

```
typedef struct {            /* UART0 Structure       */
  __IO uint32_t  DR;        /* UART Data             */
  union {
    __IO uint32_t  ECR_UART_ALT; /* UART Receive Status/Error Clear   */
    __IO uint32_t  RSR;          /* UART Receive Status/Error Clear   */
    };
  __I  uint32_t  RESERVED[4];
  __IO uint32_t  FR;            /* UART Flag                         */
  __I  uint32_t  RESERVED1;
  __IO uint32_t  ILPR;          /* UART IrDA Low-Power Register      */
  __IO uint32_t  IBRD;          /* UART Integer Baud-Rate Divisor    */
  __IO uint32_t  FBRD;          /* ART Fractional Baud-Rate Divisor  */
  __IO uint32_t  LCRH;          /* UART Line Control                 */
  __IO uint32_t  CTL;           /* UART Control                      */
  __IO uint32_t  IFLS;          /* UART Interrupt FIFO Level Select  */
  __IO uint32_t  IM;            /* UART Interrupt Mask               */
  __IO uint32_t  RIS;           /* UART Raw Interrupt Status         */
  __IO uint32_t  MIS;           /* UART Masked Interrupt Status      */
  __O  uint32_t  ICR;           /* UART Interrupt Clear              */
  __IO uint32_t  DMACTL;        /* UART DMA Control                  */
  __I  uint32_t  RESERVED2[22];
  __IO uint32_t  _9BITADDR;     /* UART 9-Bit Self Address           */
  __IO uint32_t  _9BITAMASK;    /* UART 9-Bit Self Address Mask      */
  __I  uint32_t  RESERVED3[965];
  __IO uint32_t  PP;            /* UART Peripheral Properties        */
  __I  uint32_t  RESERVED4;
  __IO uint32_t  CC;            /* UART Clock Configuration          */
} UART0_Type;
```

UART Data Structure

[file: TM4C123GH6PM7.h]

```
  /** UART memory map  **/
  #define UART0_BASE                0x4000C000UL
  #define UART1_BASE                0x4000D000UL
  #define UART2_BASE                0x4000E000UL
  #define UART3_BASE                0x4000F000UL
  #define UART4_BASE                0x40010000UL
  #define UART5_BASE                0x40011000UL
  #define UART6_BASE                0x40012000UL
  #define UART7_BASE                0x40013000UL

  /** UART declaration   **/
  #define UART0       ((UART0_Type  *) UART0_BASE)
  #define UART1       ((UART0_Type  *) UART1_BASE)
  #define UART2       ((UART0_Type  *) UART2_BASE)
  #define UART3       ((UART0_Type  *) UART3_BASE)
  #define UART4       ((UART0_Type  *) UART4_BASE)
  #define UART5        ((UART0_Type  *) UART5_BASE)
  #define UART6       ((UART0_Type  *) UART6_BASE)
  #define UART7       ((UART0_Type  *) UART7_BASE)
```

```
/** accessing UART registers  **/
UART0->DR = 0x14;
UART1->CTL |= 1UL <<4    /* set bit 4  */
UART1->IBRD = 0x05;
```

UART Initialization (*Example*)

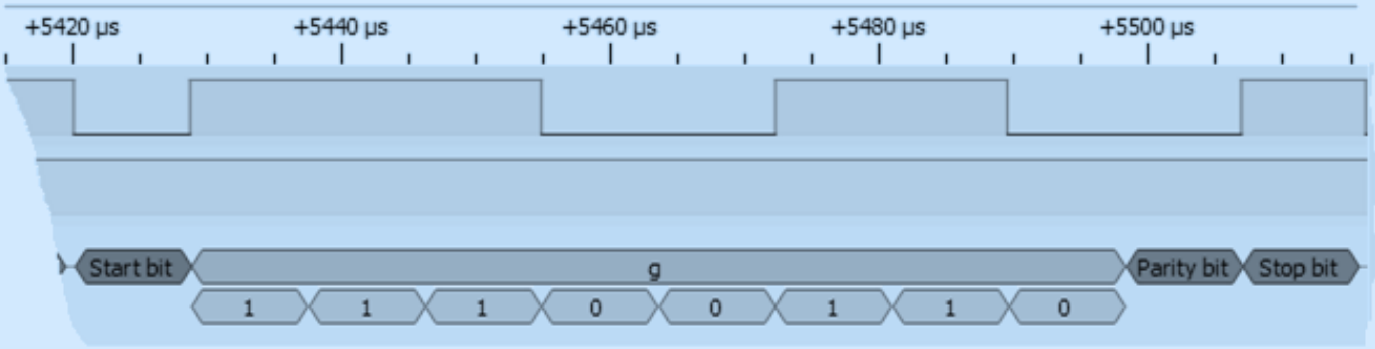
```
void UART_Init(void)
{
   SYSCTL->RCGCUART |= SYSCTL_RCGCUART_R1; /* 1. enable UART1 clock       */
   /* wait for UART1 to be ready  */
   while( 0 == (SYSCTL->PRUART & SYSCTL_PRUART_R1)) {};
   UART1->CTL &= ~UART_CTL_UARTEN;     /* 3. disable UART during init  */
   UART1->CTL &= ~UART_CTL_HSE;        /* 4. use 16X CLKDIV            */

   /** set baud rate - 921,600 bps   **/
   UART1->IBRD = 5;  /* 4. int(80,000,000/(16*921,600))=int(5.42534)=5 */
   UART1->FBRD = 27; /* 5. int(0.42535*64 + 0.5=int(27.72)=27          */

   /** 6. write to LCRH reg  **/
   UART1->LCRH &= ~UART_LCRH_WLEN_M;
   UART1->LCRH |= UART_LCRH_WLEN_8;    /* 8 data bits                  */
   UART1->LCRH &= ~UART_LCRH_STP2;     /* 1 stop bit                   */
   UART1->LCRH &= ~UART_LCRH_FEN;      /* disable FIFO                 */

   UART1->CC &= ~UART_CC_CS_M;         /* 7. UART clock source         */
   UART1->CC |= UART_CC_CS_SYSCLK;     /*    set to System Clock       */
   UART1->CTL |= UART_CTL_TXE||UART_CTL_RXE; /* 8. Tx & Rx enable      */
   UART1->CTL |= UART_CTL_UARTEN;      /* 8. enable UART1              */
}
```

Review Questions - UART *(Week 5)*

1. Explain what you understand by half duplex and full duplex serial communications.
2. Explain how parity works during UART communications.
3. How many UARTs are implemented on the Tiva LaunchPad? Name them.
4. What advantage(s) does a FIFO offer in UART communications?
5. What is the data width of the Transmit FIFO?
6. Why is the data width of the Receive FIFO larger than that of the Transmit FIFO? What are the extra bits for?
7. At a system clock of 80 MHz, how can you program UART1 on the LaunchPad to provide a baud rate of 115,200 bits/s?
8. For the above implementation, what is the error rate?
9. Write a short program to initialize the relevant GPIO pins to function as the Transmit and Receive pins for UART1 on the LaunchPad. Assume polling is used.
10. Which register is used to set the TX FIFO trigger level for a UART? What value should be written to the register to set the RX FIFO to trigger at ¼ full and the TX FIFO to trigger at ¼ full?

Review Questions - UART *(Week 5)*

11. Which register is used to set the UART Baud Rate clock to use the PIOSC clock source? What value should you write to the register?
12. The figure below shows a UART data frame with LB bit transmitted first.
    - What is the ASCII character ('DATA') sent by the UART?
    - Is the data sent with Even or Odd parity? Explain your reasoning.



Note ….

- *Please email me at foohoong.fong@digipen.edu if you find any typos or errors in the lecture notes.*