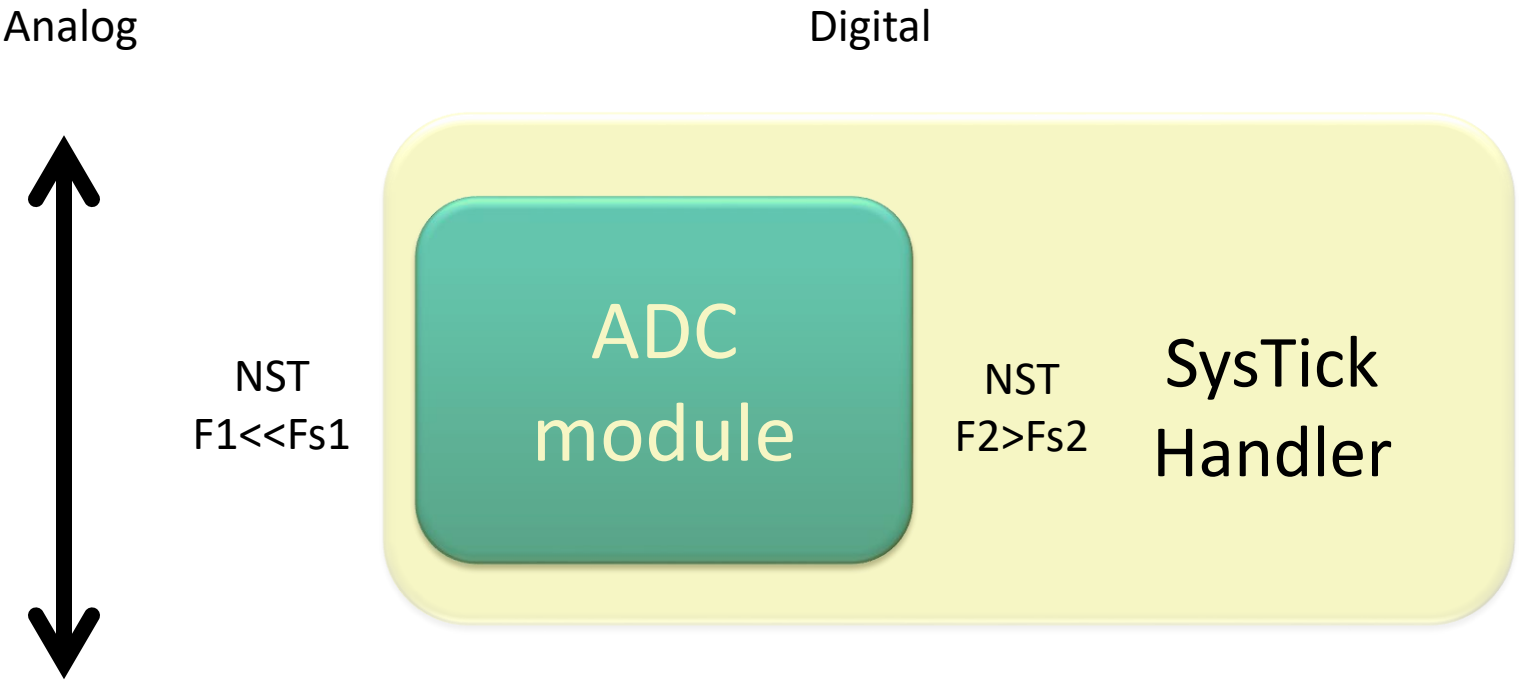
CET 241: Day 15
UART

Dr. Noori Kim

ADC lab and NST



Speed of My hand
1-2Hz (??)

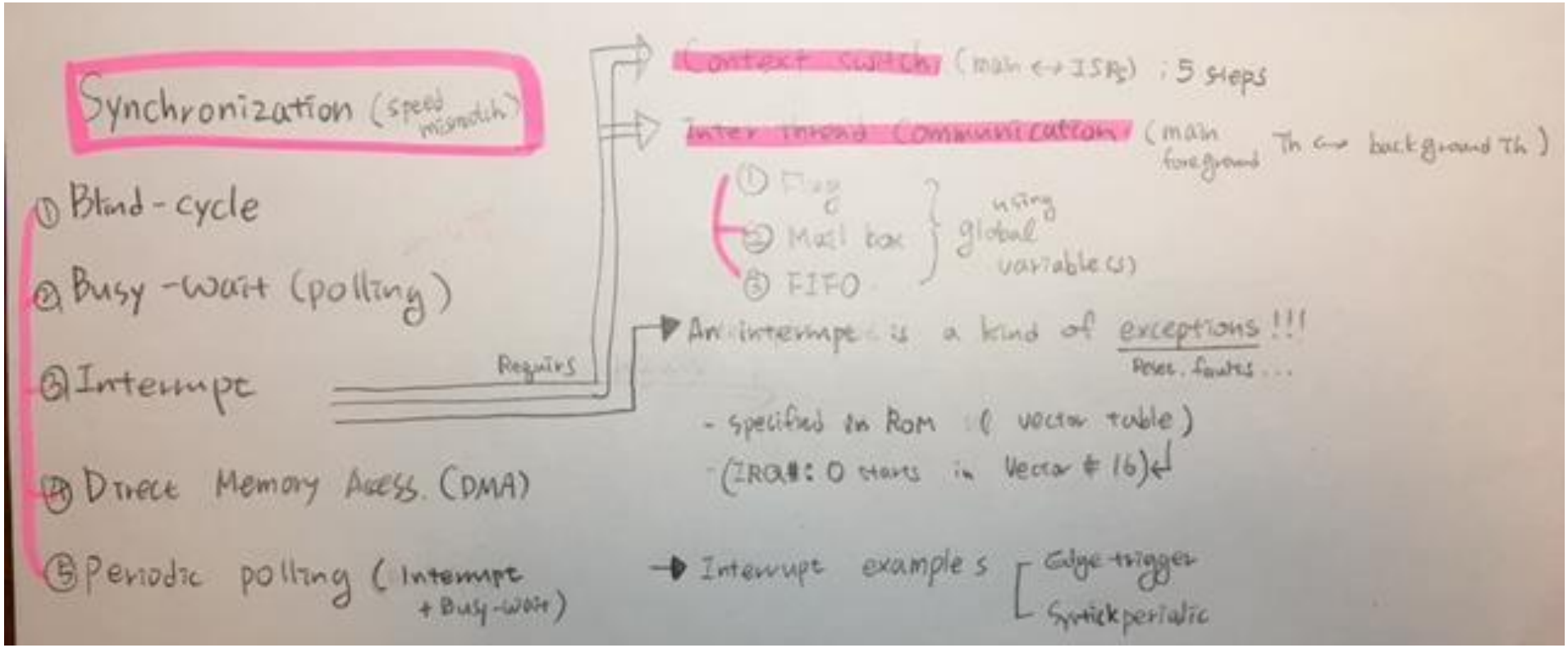
Speed of ADC
125KHz

Speed of SysTick
80MHz MCU/Reload value 1500
=53KHz

```
void SysTick_Handler(void){
  Distance = Convert(ADC0_In());
}
```

Overall NST
F1<<Fs2

A big picture



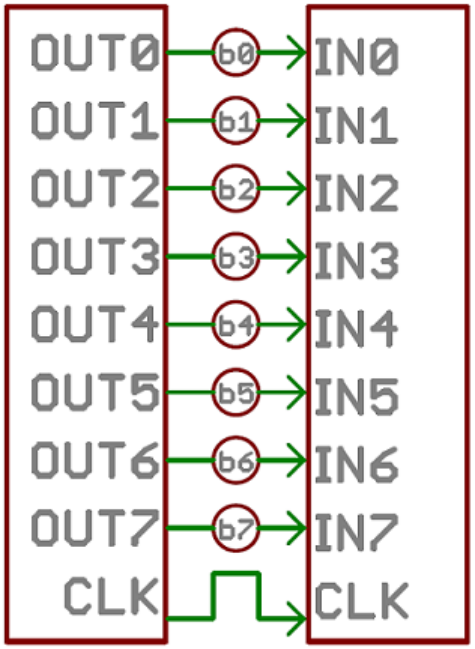
What is the Synchronization?

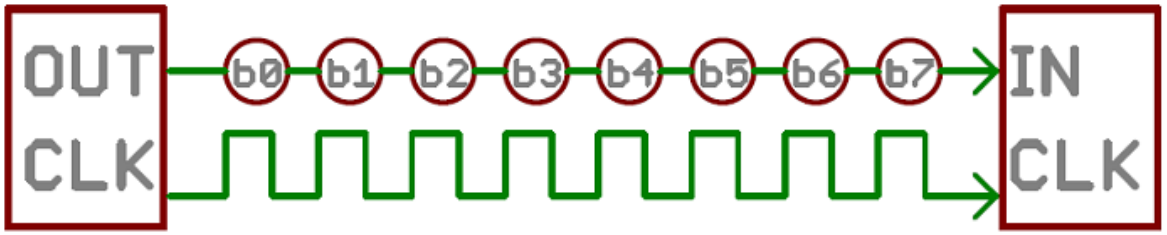
- When **connecting two components** together, we need a way to **Synchronize** them
  - Components include both software and hardware
- To compensate (or bridge) **speed mismatch** of communicating devices
  - A is too fast compared to B
  - Then, A is waiting for B until B's task is done

- We’ve talked about general synchronization methods.
  - Actually we’ve used these but just did not appreciate the concept.
- Having these concepts in our mind,
  - We will learn about one of serial interfaces, UART.
  - We will apply busy-wait (polling) synchronization on the UART example.

Serial Communication: (With the Busy-wait Synch. UART example)

- Parallel interfaces: transferring multiple bits at the same time
  - Fast, straightforward, and relatively easy to implement
  - But requiring many I/O lines
- Serial interfaces: streaming their data, **one single bit at a time**.
  - Operating on as little as one wire, usually never more than four.





Serial communication in general

- Synchronous or asynchronous: clock data are transmitted or not
- Basic packet structure (or frame, the smallest complete unit of serial transmission):
  - Start bit
  - Data bits
  - Parity bits: indicating data packet error, optional
  - Stop bits

Synchronization bits

A serial frame. Some symbols in the frame have configurable bit sizes

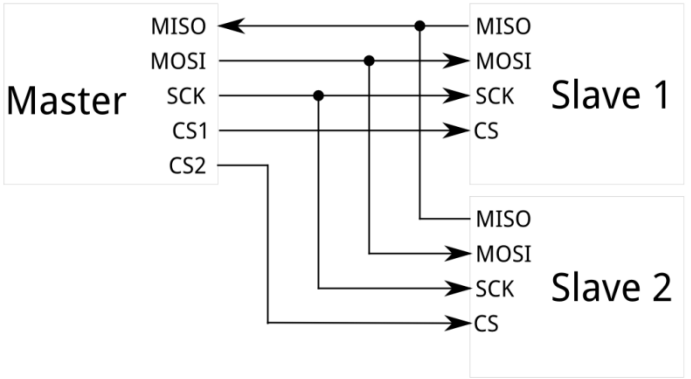
- **Baud rate**: specifying **how fast** data is sent over a serial line (units: bits-per-second, bps)
  - Inverting the baud rate: how long it takes to transmit a single bit
  - Both devices operate at the same baud rate
  - Common baud rates: **9600 bps**
    - Other "standard" s: 1200, 2400, 4800, 19200, 38400, 57600, and 115200 (max, above this: many errors)
  - The higher a baud rate goes, the faster data is sent/received
  - The reciprocal of the baud rate is called the **bit time**

- Serial interface examples
  - SPI
  - I^2C
  - **UART**

Just check How they look like

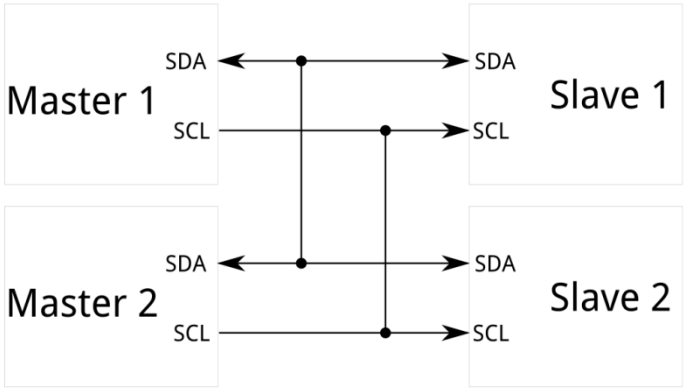
Example 1: SPI (Serial Peripheral Interface)

- Synchronous (using separate lines for data and a "clock" that keeps both sides in perfect sync)
- Connecting a single master to each slave with an SPI bus requires four lines
- Good for high data rate full-duplex (simultaneous sending and receiving of data) connections



Example 2: I²C (Inter-Integrated Circuit)

- Invented by Philips Semiconductor (now NXP Semiconductors)
- Typically used for lower-speed peripheral, short distance communications
- Allowing a multi-master system,
- Requiring two signal wires to exchange information
- Synchronous

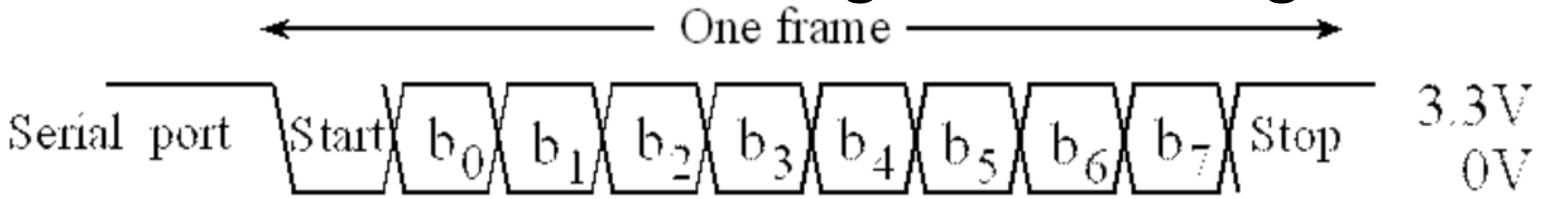


More complex communication protocol than with a UART or SPI solution

Example 3: UART in general
(A universal asynchronous receiver/ transmitter)

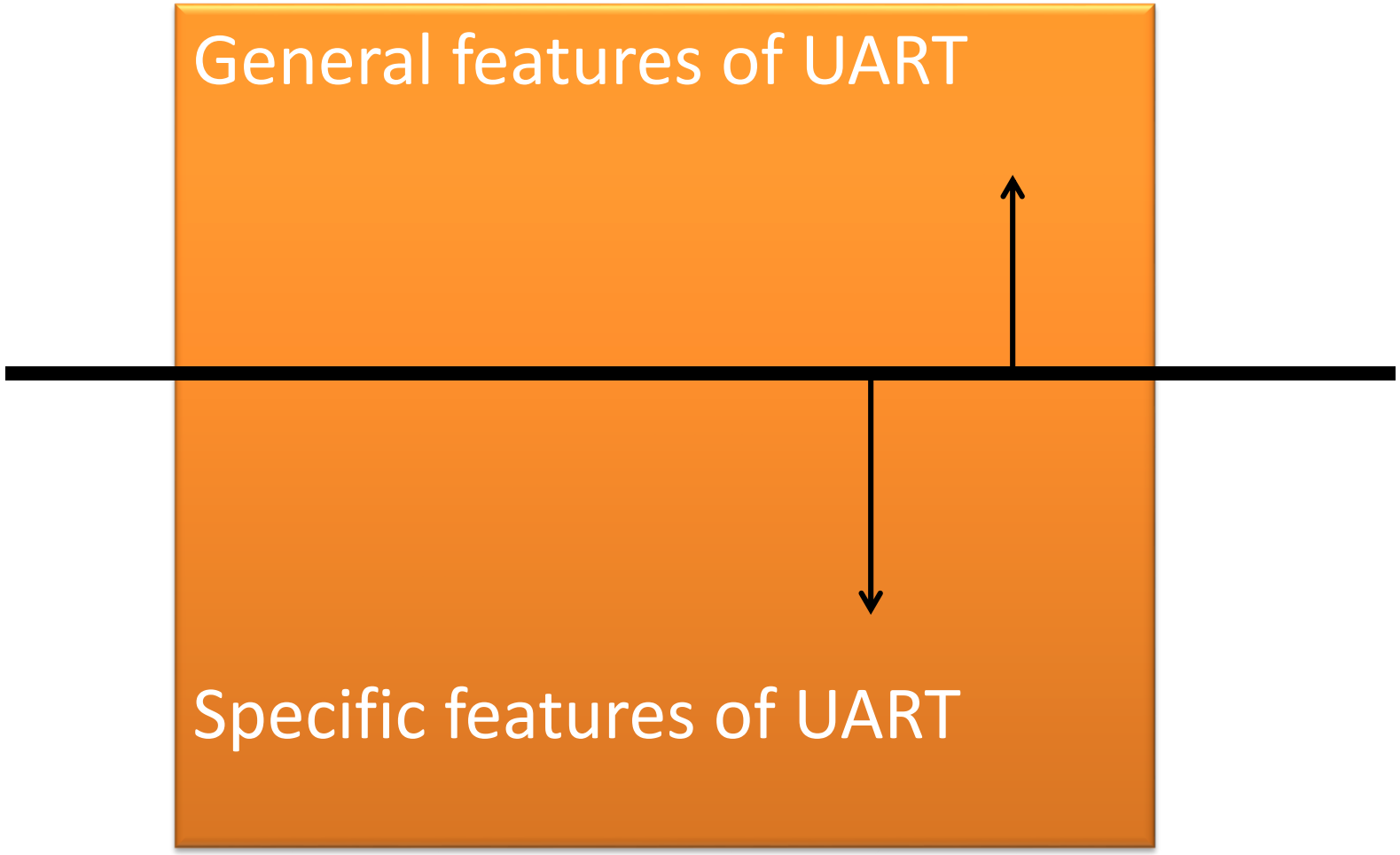
- Most microcontrollers have at least one UART
  - The LM4F120/TM4C123: 8 UARTs
  - A baud rate control register: to select the transmission rate
- **A single frame:** a start bit (which is 0), 8 bits of data, Parity bits, and a stop bit (which is 1)
  - Always only one start bit
  - The Stellaris® UARTs: to select the 5 to 8 data bits and 1 or 2 stop bits
  - Parity bit: even, odd, or no

- The typical protocol
  - 1 start bit, 8 data bits, no parity, and 1 stop bit.
  - Used for both transmitting and receiving



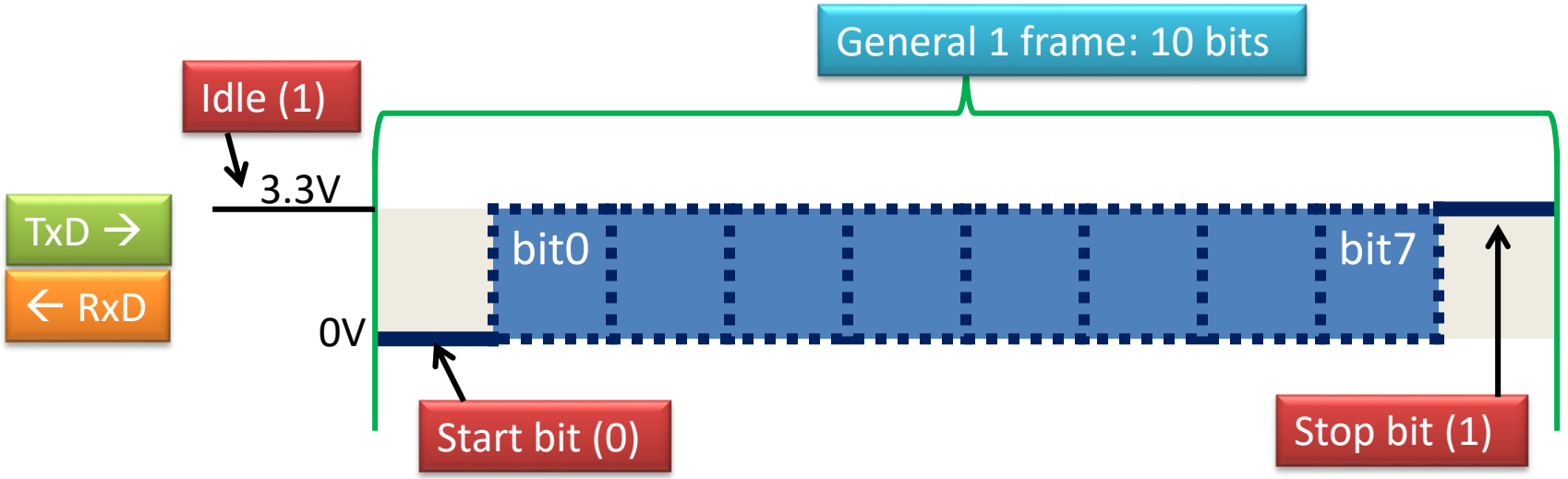
A serial data frame with 8-bit data, 1 start bit, 1 stop bit, and no parity bit.

- Bandwidth (in bytes/second)
  - The amount of data or useful information transmitted per second: information rate.
  - The baud rate (in bits/sec) divided by 10.
- Only **TxD**, **RxD**, and **SG (signal ground)** are required to implement a simple bidirectional serial channel


General features of UART
Specific features of UART

UART in specific

1. Data in serial manner: one bit/time
2. Time per a bit: Bit time $\Delta t = \frac{1}{Baud\ rate}$
3. Output: transmitter (TxD)
4. Input: receiver (RxD)



There are two parts in UART

- Transmission
- Receiver

- Features of the Stellaris UART:
  - A 16x8 bit (or 16x12) receive FIFO and a 16x8 bit transmit FIFO

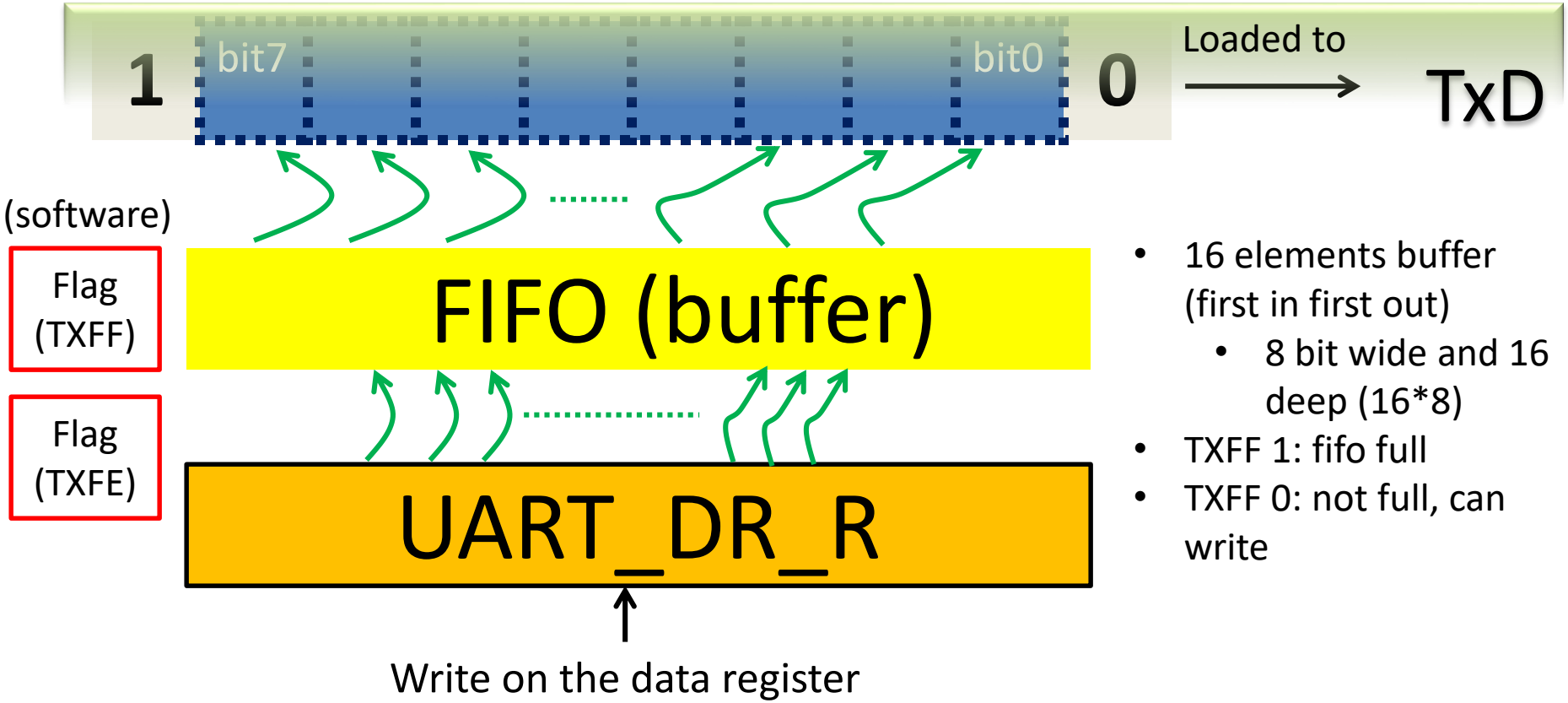
14.3.8 FIFO Operation

Page900

The UART has two 16x8 FIFOs; one for transmit and one for receive. Both FIFOs are accessed via the **UART Data (UARTDR)** register (see page 906). Read operations of the **UARTDR** register return a 12-bit value consisting of 8 data bits and 4 error flags while write operations place 8-bit data in the transmit FIFO.

Transmitter side

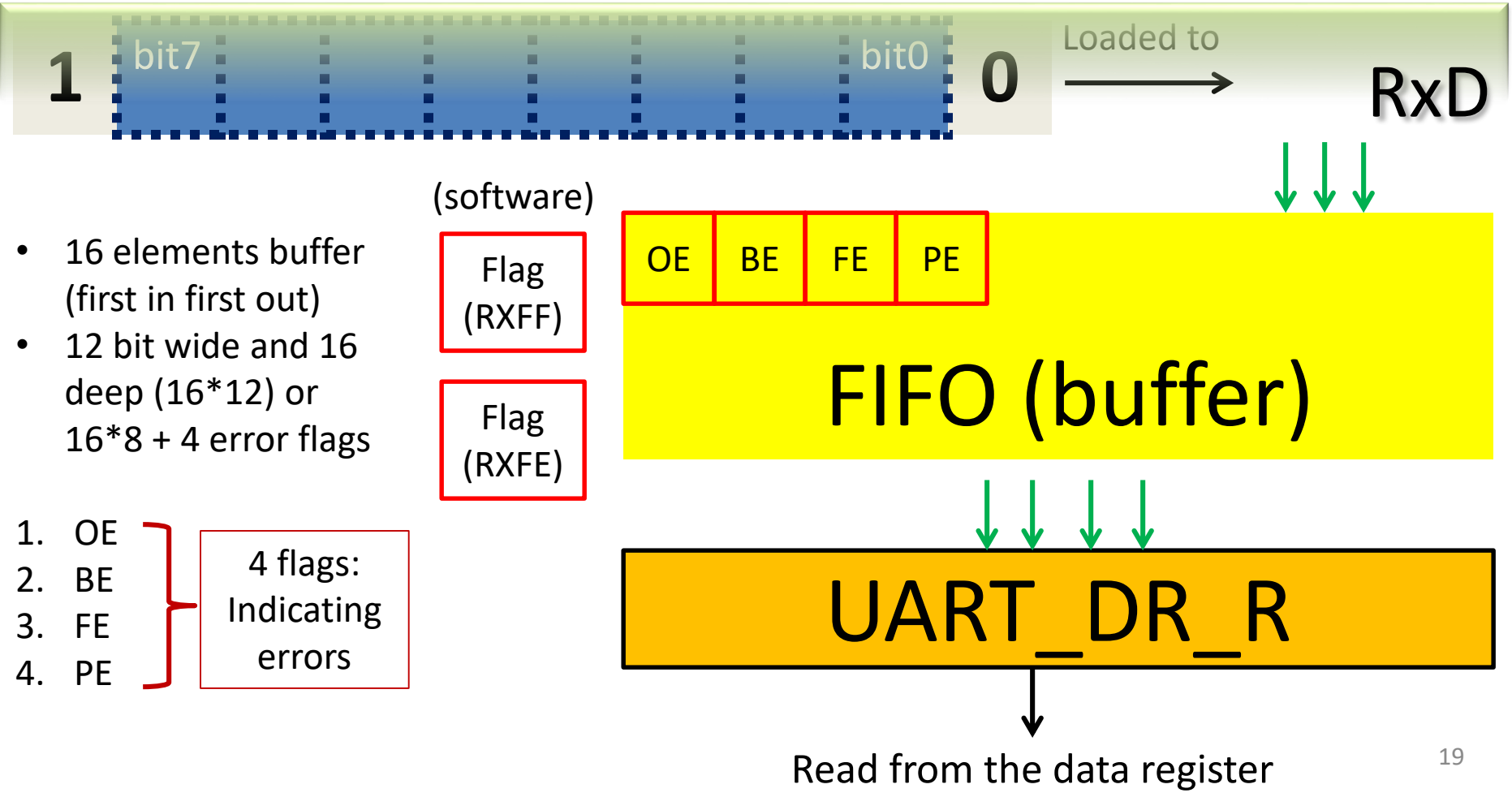
- TxD: 10-bit shift register



- 16 elements buffer (first in first out)
  - 8 bit wide and 16 deep (16*8)
- TXFF 1: fifo full
- TXFF 0: not full, can write

Receiver side

- RxD: 10-bit shift register



- 16 elements buffer (first in first out)
- 12 bit wide and 16 deep (16*12) or 16*8 + 4 error flags

1. OE
2. BE
3. FE
4. PE

4 flags: Indicating errors

OE: over run error

When it is 1,

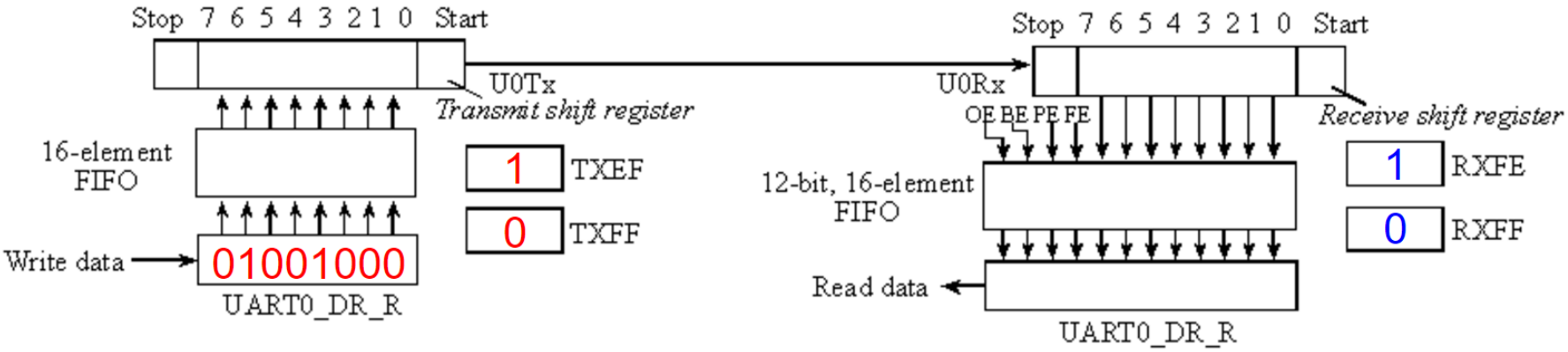
- New data was received when the FIFO was full, resulting in data loss
- a receiver speaks to a sender (transmitter)
  - Receiver buffer is full
  - Some frames have to be dropped
  - You, the sender is sending too fast

Software side using a different flag, **RXFE flag** (receive buffer FIFO empty)

- The receiver repeatedly check this flag
- 1: no data to be consumed in FIFO (busy-wait)
- 0: FIFO not empty, some data to be consumed

- BE: Break error (ignore, rarely used)
  - receive data input was held Low for longer than a full-word transmission time (defined as start, data, parity, and stop bits).
- FE: frame error (Baud-rate error),
  - **Framing errors** normally occur with **baud rate** mismatches; the sender and receiver (S<R) have not been configured with the same Baud-rate.
  - The received character does not have a valid stop bit (a valid stop bit is 1).
- PE: parity error, no longer use it, just there for continuity in the protocol

How the UART works (video)



ASCII codes for “H” and “i”?

72, 105

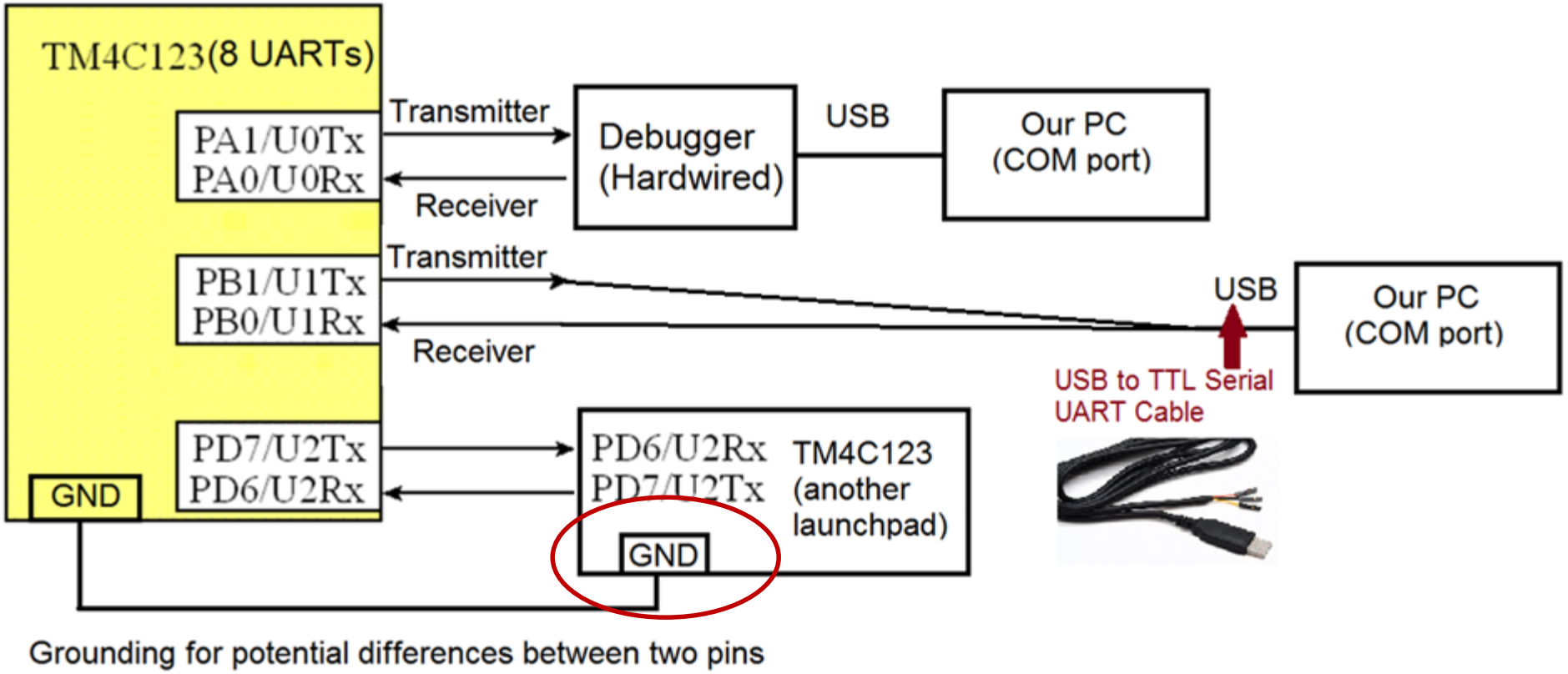
- FIFO FULL (FF) vs FIFO EMPTY (FE)?
  - FF=1 equals to FE=0?
- Obviously quite different meanings
  - TxFE, TxFF both 1: this would indicate both full and empty and thus is **not a legal state**.

- Describe what happens if the receiving computer is operating on a baud rate that is twice as fast as **the transmitting computer**?
  - The data will be received in error (values will not be correct).
  - The receiver **could appear** to get two input frames for every one frame transmitted (as its baud rate is faster).
  - It will probably cause framing errors (FE).
  - It would cause parity errors if active.
  - For sure that it's not OE

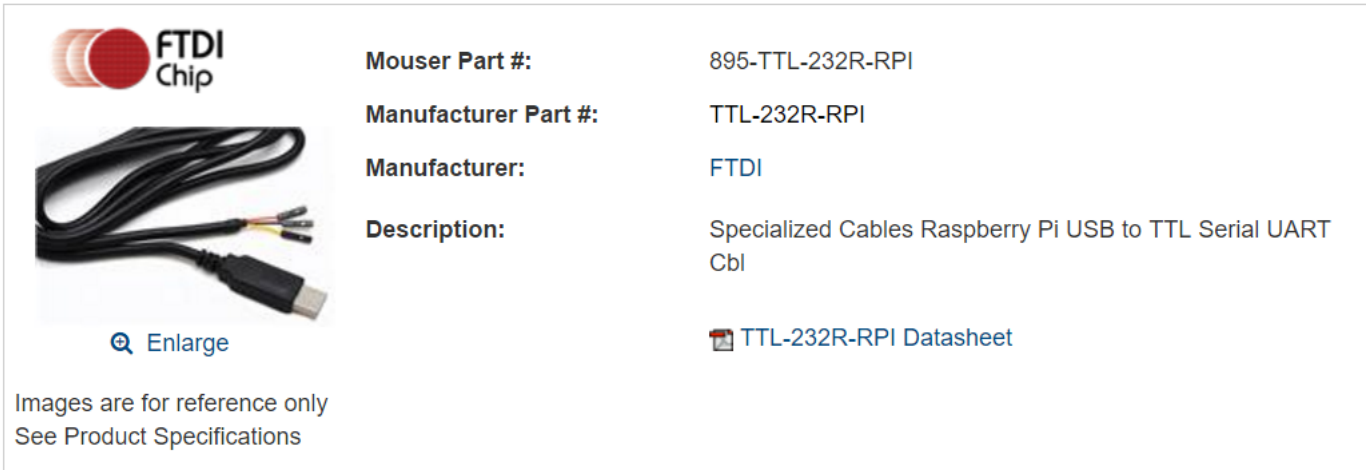
- Describe what happens if the transmitting computer is operating on a baud rate that is twice as fast as **the receiving computer**?
  - The data will be received in error (values will not be correct).
  - The receiver **will appear** to get one input frame for every two frame transmitted.
  - It will probably not cause framing errors (FE).
  - It would cause OE and PE if active.

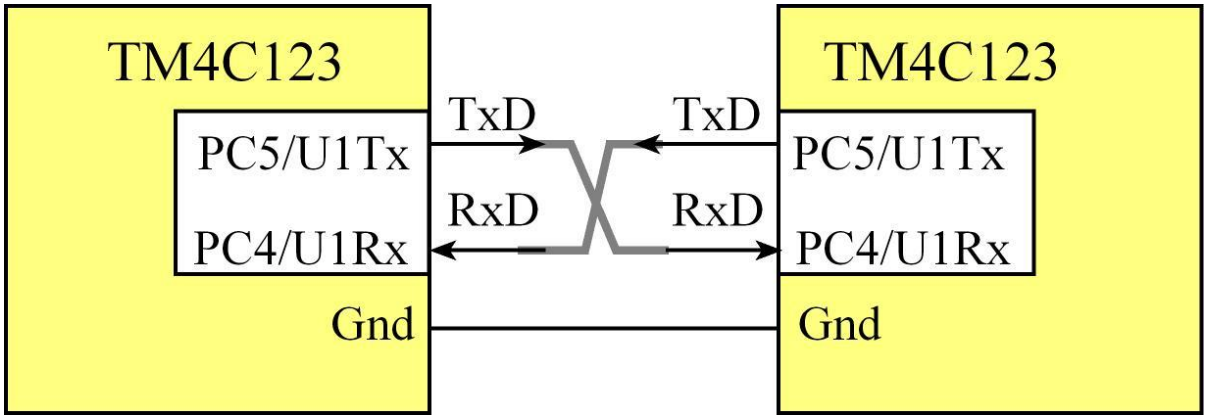
Checkpoint 4.13: The data will be received in error (values will not be correct). The receiver will appear to get one input frame for every one frame transmitted. It will probably not cause framing errors (FE). It would cause parity errors if active.

UART on TM4C123 interfacing examples



- http://www.mouser.sg/ProductDetail/FTDI/TTL-232R-RPI/?qs=3tuk1l7PSbNqj0mOeA5IPw%3d%3d





Simple serial interface between two microcontrollers.

TX and RX Must be crossed

- We’ve talked about many general things about UART so far. Our mission is to find out registers and bits to apply this concept to implement UART in our MCU.

TM4C UART Details

- Turn on the UART clock in the **RCGC1** register.
- Turn on the clock for the digital port in the **RCGC2** register.
- Enable the transmit and receive pins as digital signals (DEN).
- Alternative function must also be selected as UART, set bit in PCTL registers.
- The OE, BE, PE, and FE are error flags associated with the receiver. You can see these flags in two places
  - associated with each data byte in **UART0_DR_R** or
  - as a separate error register in **UART0_RSR_R**.

UART0 registers

	31–12	11	10	9	8	7–0	Name
$4000.C000		OE	BE	PE	FE	DATA	UART0_DR_R

DR: Data Register

	31–3	3	2	1	0	
$4000.C004		OE	BE	PE	FE	UART0_RSR_R

RSR: Receive Status Register

	31–8	7	6	5	4	3	2–0	
$4000.C018		TXFE	RXFF	TXFF	RXFE	BUSY		UART0_FR_R

FR: Flag Register

	31–16	15–0	
$4000.C024		DIVINT	UART0_IBRD_R

IBRD: Integer Baud Rate Divisor

	31–6	5–0	
$4000.C028		DIVFRAC	UART0_FBRD_R

FBRD: Fraction Baud Rate Divisor

$4000.C02C		SPS	WPEN	FEN	STP2	EPS	PEN	BRK	UART0_LCRH_R

LCRH: Line ContRol High byte

	31–10	9	8	7	6–3	2	1	0	
$4000.C030		RXE	TXE	LBE		SIRLP	SIREN	UARTEN	UART0_CTL_R

CTL: ConToL

Page900 datasheet

14.3.8 FIFO Operation

The UART has two 16x8 FIFOs; one for transmit and one for receive. Both FIFOs are accessed via the **UART Data (UARTDR)** register (see page 906). Read operations of the **UARTDR** register return a 12-bit value consisting of 8 data bits and 4 error flags while write operations place 8-bit data in the transmit FIFO.

Out of reset, both FIFOs are disabled and act as 1-byte-deep holding registers. The FIFOs are enabled by setting the FEN bit in **UARTLCRH** (page 916).

FIFO status can be monitored via the **UART Flag (UARTFR)** register (see page 911) and the **UART Receive Status (UARTRSR)** register. Hardware monitors empty, full and overrun conditions. The **UARTFR** register contains empty and full flags (TXFE, TXFF, RXFE, and RXFF bits), and the **UARTRSR** register shows overrun status via the OE bit. If the FIFOs are disabled, the empty and full flags are set according to the status of the 1-byte-deep holding registers.

The trigger points at which the FIFOs generate interrupts is controlled via the **UART Interrupt FIFO Level Select (UARTIFLS)** register (see page 922). Both FIFOs can be individually configured to trigger interrupts at different levels. Available configurations include ⅛, ¼, ½, ¾, and ⅞. For example, if the ¼ option is selected for the receive FIFO, the UART generates a receive interrupt after 4 data bytes are received. Out of reset, both FIFOs are configured to trigger an interrupt at the ½ mark.

- The status of the two FIFOs can be seen in the **UART0_FR_R** register.
- The **BUSY** flag is set while the transmitter still has unsent bits, even if the transmitter is disabled
  - zero when the transmit FIFO is empty and the last stop bit has been sent)

UART Flag (UARTFR), offset 0x018

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								TXFE	RXFF	TXFF	RXFE	BUSY	reserved		CTS
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	1	0	0	1	0	0	0	0

3	BUSY	RO	0	UART Busy

Value	Description
0	The UART is not busy.
1	The UART is busy transmitting data. This bit remains set until the complete byte, including all stop bits, has been sent from the shift register.

This bit is set as soon as the transmit FIFO becomes non-empty (regardless of whether UART is enabled).

- The **UART0_CTL_R** control register contains the bits that turn on the UART.
  - **TXE** is the Transmitter Enable bit
  - **RXE** is the Receiver Enable bit.
  - Set **TXE**, **RXE**, and **UARTEN** equal to 1 in order to activate the UART device.
  - Clear **UARTEN** during the initialization sequence.

- The **IBRD** and **FBRD** registers specify the baud rate.
- The baud rate **divider** is a 22-bit binary fixed-point value with a resolution of 2^{-6}
- The **Baud16** clock is created from the system bus clock, with a frequency of (Bus clock frequency)/**divider**. The baud rate is 16 times slower than **Baud16**
- **Baud rate** = **Baud16**/**16** = (Bus clock frequency)/(16*__divider__)

- For example, if the bus clock is 80 MHz and the desired baud rate is 19200 bits/sec, then the **divider** should be 80,000,000/16/19200 or 260.4167.
  - Let *m* be the integer part, without rounding. We store the integer part (*m*=260) in **IBRD**.
  - For the fraction, we find an integer *n*, such that *n*/64 is about 0.4167. More simply, we multiply 0.4167*64 = 26.6688 and round to the closest integer, 27. We store this fraction part (*n*=27) in **FBRD**.

- Determine the actual baud rate. Assume the bus clock is 80 MHz, when **IBRD** =260 and **FBRD** =27
  - **Baud rate** = (80 MHz)/(16* (*m*+*n*/64)) = (80 MHz)/(16* (260+27/64)) = 19199.616 bits/sec
- The original Baud rate was 19200
  - The baud rates in the transmitter and receiver must match **within 5%** for the channel to operate properly.

- **LCRH**, **IBRD**, and **FBRD** form an internal 30-bit register.
  - This internal register is only updated when a write operation to **LCRH** is performed
  - Any changes to the baud-rate divisor must be followed by a write to the **LCRH** register for the changes to take effect
  - The FIFOs are enabled by setting the **FEN** bit in **LCRH**.

50MHz, 115200 baud rate ex.

- IBRD=int(50000000/(16*115,200))= int(27.1267)=27
- FBRD = round(0.1267 * 64) = 8

Page 902 datasheet :Copy and paste and fill-in each step followed by instruction

```
void UART_Init(void){
    // 1. Enable the UART module using the RCGCUART register (see page 344).
    // 2. Enable the clock to the appropriate GPIO module via the RCGCGPIO register (see page 340).
    // To find out which GPIO port to enable, refer to Table 23-5 on page 1351.

    // 3. Set the GPIO AFSEL bits for the appropriate pins (see page 671). To determine which GPIOs to
    // configure, see Table 23-4 on page 1344

    // 4. Configure the GPIO current level and/or slew rate as specified for the mode selected (see
    // page 673 and page 681, not reqired for this example

    // 5. Configure the PMCn fields in the GPIOPCTL register to assign the UART signals to the appropriate
    // pins (see page 688 and Table 23-5 on page 1351).

    // With the BRD values in hand, the UART configuration is written to the module in the following order
    // 1. Disable the UART by clearing the UARTEN bit in the UARTCTL register
       // Find  the Baud-Rate Divisor; baud-rate: 115200
                  // IBRD = int(80,000,000 / (16 * 115200))
                  // FBRD = round(0.402778 * 64)
    // 2. Write the integer portion of the BRD to the UARTIBRD register
    // 3. Write the fractional portion of the BRD to the UARTFBRD register.
    // 4. Write the desired serial parameters to the UARTLCRH register
    // 5. Configure the UART clock source by writing to the UARTCC register
    // 6. Optionally, configure the µDMA channel (see "Micro Direct Memory Access (µDMA)" on page 585)
    // and enable the DMA option(s) in the UARTDMACTL register
    // 7. Enable the UART by setting the UARTEN bit in the UARTCTL register.
}
```

// 1. Enable the UART module using the RCGCUART register (see page 344).

Register 63: Universal Asynchronous Receiver/Transmitter Run Mode Clock Gating Control (RCGCUART), offset 0x618

The **RCGCUART** register provides software the capability to enable and disable the UART modules in Run mode. When enabled, a module is provided a clock and accesses to module registers are allowed. When disabled, the clock is disabled to save power and accesses to module registers generate a bus fault. This register provides the same capability as the legacy **Run Mode Clock Gating Control Register n RCGCn** registers specifically for the watchdog modules and has the same bit polarity as the corresponding **RCGCn** bits.

Important: This register should be used to control the clocking for the UART modules. To support legacy software, the **RCGC1** register is available. A write to the **RCGC1** register also writes the corresponding bit in this register. Any bits that are changed by writing to the **RCGC1** register can be read back correctly with a read of the **RCGC1** register. Software

	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

0	R0	RW	0	UART Module 0 Run Mode Clock Gating Control

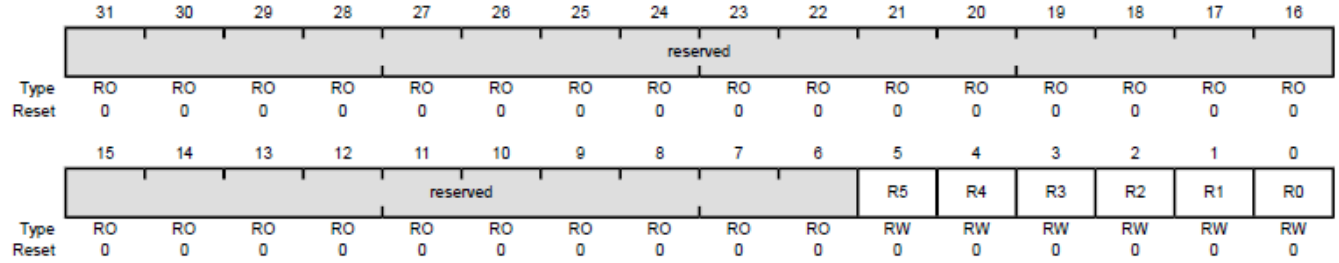
Value	Description
0	UART module 0 is disabled.
1	Enable and provide a clock to UART module 0 in Run mode.

// 2. Enable the clock to the appropriate GPIO module via the RCGCGPIO register (see page 340). // To find out which GPIO port to enable, refer to Table 23-5 on page 1351.

Register 60: General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO), offset 0x608

The **RCGCGPIO** register provides software the capability to enable and disable GPIO modules in Run mode. When enabled, a module is provided a clock and accesses to module registers are allowed. When disabled, the clock is disabled to save power and accesses to module registers generate a bus fault. This register provides the same capability as the legacy **Run Mode Clock Gating Control Register n RCGCn** registers specifically for the watchdog modules and has the same bit polarity as the corresponding **RCGCn** bits.

Important: This register should be used to control the clocking for the GPIO modules. To support legacy software, the **RCGC2** register is available. A write to the **RCGC2** register also



0	R0	RW	0	GPIO Port A Run Mode Clock Gating Control

Value	Description
0	GPIO Port A is disabled.
1	Enable and provide a clock to GPIO Port A in Run mode.

// 3. Set the GPIO AFSEL bits for the appropriate pins (see page 671). To determine which GPIOs to configure, see Table 23-4 on page 1344

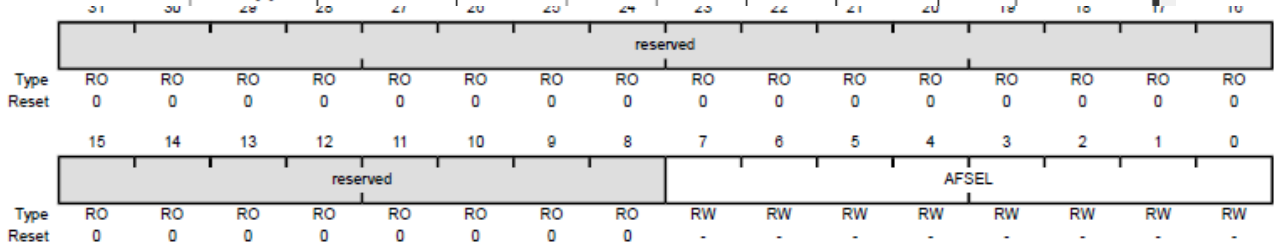
Register 10: GPIO Alternate Function Select (GPIOAFSEL), offset 0x420

The **GPIOAFSEL** register is the mode control select register. If a bit is clear, the pin is used as a GPIO and is controlled by the GPIO registers. Setting a bit in this register configures the corresponding GPIO line to be controlled by an associated peripheral. Several possible peripheral functions are multiplexed on each GPIO. The **GPIO Port Control (GPIOPCTL)** register is used to select one of the possible functions. Table 23-5 on page 1351 details which functions are muxed on each GPIO pin. The reset value for this register is 0x0000.0000 for GPIO ports that are not listed in the table below.

Important: The table below shows special consideration GPIO pins. Most GPIO pins are configured as GPIOs and tri-stated by default (**GPIOAFSEL**=0, **GPIODEN**=0, **GPIOPDR**=0, **GPIOPUR**=0, and **GPIOPCTL**=0). Special consideration pins may be programed to a non-GPIO function or may have special commit controls out of reset. In addition, a Power-On-Reset (POR) or asserting RST returns these GPIO to their original special consideration state.

Table 10-7. GPIO Pins With Special Considerations

GPIO Pins	Default Reset State	GPIOAFSEL	GPIODEN	GPIOPDR	GPIOPUR	GPIOPCTL	GPIOCR
PA[1:0]	UART0	0	0	0	0	0x1	1
PA[5:2]	SSI0	0	0	0	0	0x2	1
PB[3:2]	I[21]C0	0	0	0	0	0x3	1
PC[3:0]	JTAG/SWD	1	1	0	1	0x1	0
PD[7]	GPIO[a]	0	0	0	0	0x0	0
PF[0]	GPIO[a]	0	0	0	0	0x0	0



7:0	AFSEL	RW	-	GPIO Alternate Function Select

Value	Description
0	The associated pin functions as a GPIO and is controlled by the GPIO registers.
1	The associated pin functions as a peripheral signal and is controlled by the alternate hardware function. The reset value for this register is 0x0000.0000 for GPIO ports that are not listed in Table 10-1 on page 650.

// 4. Configure the GPIO current level and/or slew rate as specified for the mode selected (see page 673 and page 681,
// not required for this example

// 5. Configure the PMCn fields in the GPIOPCTL register to assign the UART signals to the appropriate pins (see page 688 and Table 23-5 on page 1351).
//I am sure that you can do this

// With the BRD values in hand, the UART configuration is written to the module in the following order

// 1. Disable the UART by clearing the UARTEN bit in the UARTCTL register

Register 8: UART Control (UARTCTL), offset 0x030

The **UARTCTL** register is the control register. All the bits are cleared on reset except for the Transmit Enable (TXE) and Receive Enable (RXE) bits, which are set.

To enable the UART module, the UARTEN bit must be set. If software requires a configuration change in the module, the UARTEN bit must be cleared before the configuration changes are written. If the UART is disabled during a transmit or receive operation, the current transaction is completed prior to the UART stopping.

Note: The **UARTCTL** register should not be changed while the UART is enabled or else the results are unpredictable. The following sequence is recommended for making changes to the **UARTCTL** register.

1. Disable the UART.
2. Wait for the end of transmission or reception of the current character.
3. Flush the transmit FIFO by clearing bit 4 (FEN) in the line control register (**UARTLCRH**).
4. Reprogram the control register.
5. Enable the UART.

	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

0	UARTEN	RW	0	UART Enable

Value	Description
0	The UART is disabled.
1	The UART is enabled.

If the UART is disabled in the middle of transmission or reception, it completes the current character before stopping.

// Find the Baud-Rate Divisor; baud-rate: 115200

// IBRD = int(Bus Clock/ (16 * baud-rate))

// FBRD = round(floating remainder * 64)

// 2. Write the integer portion of the BRD to the UARTIBRD register

// 3. Write the fractional portion of the BRD to the UARTFBRD register.

Register 5: UART Integer Baud-Rate Divisor (UARTIBRD), offset 0x024

The **UARTIBRD** register is the integer part of the baud-rate divisor value. All the bits are cleared on reset. The minimum possible divide ratio is 1 (when **UARTIBRD**=0), in which case the **UARTFBRD** register is ignored. When changing the **UARTIBRD** register, the new value does not take effect until transmission/reception of the current character is complete. Any changes to the baud-rate divisor must be followed by a write to the **UARTLCRH** register. See "Baud-Rate Generation" on page 896 for configuration details.

	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
	DIVINT															
Type	RW	RW	RW	RW	RW	RW	RW	RW	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

Register 6: UART Fractional Baud-Rate Divisor (UARTFBRD), offset 0x028

The **UARTFBRD** register is the fractional part of the baud-rate divisor value. All the bits are cleared on reset. When changing the **UARTFBRD** register, the new value does not take effect until transmission/reception of the current character is complete. Any changes to the baud-rate divisor must be followed by a write to the **UARTLCRH** register. See "Baud-Rate Generation" on page 896 for configuration details.

	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

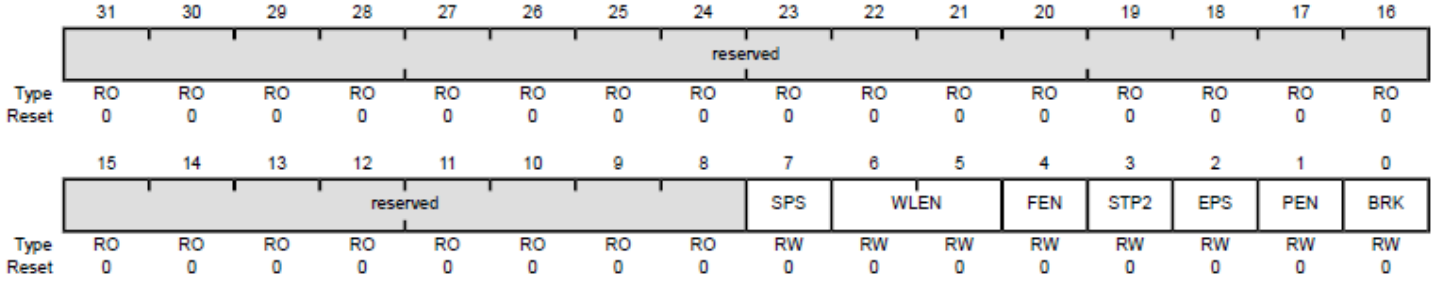
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved										DIVFRAC					
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

// 4. Write the desired serial parameters to the UARTLCRH register

Register 7: UART Line Control (UARTLCRH), offset 0x02C

The **UARTLCRH** register is the line control register. Serial parameters such as data length, parity, and stop bit selection are implemented in this register.

When updating the baud-rate divisor (**UARTIBRD** and/or **UARTIFRD**), the **UARTLCRH** register must also be written. The write strobe for the baud-rate divisor registers is tied to the **UARTLCRH** register.



7	SPS	RW	0	UART Stick Parity Select When bits 1, 2, and 7 of **UARTLCRH** are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set and 2 is cleared, the parity bit is transmitted and checked as a 1. When this bit is cleared, stick parity is disabled.
6:5	WLEN	RW	0x0	UART Word Length The bits indicate the number of data bits transmitted or received in a frame as follows: Value Description 0x0 5 bits (default) 0x1 6 bits 0x2 7 bits 0x3 8 bits

Bit/Field	Name	Type	Reset	Description
4	FEN	RW	0	UART Enable FIFOs Value Description 0 The FIFOs are disabled (Character mode). The FIFOs become 1-byte-deep holding registers. 1 The transmit and receive FIFO buffers are enabled (FIFO mode).
3	STP2	RW	0	UART Two Stop Bits Select Value Description 0 One stop bit is transmitted at the end of a frame. 1 Two stop bits are transmitted at the end of a frame. The receive logic does not check for two stop bits being received. When in 7816 smartcard mode (the SMART bit is set in the **UARTCTL** register), the number of stop bits is forced to 2.
2	EPS	RW	0	UART Even Parity Select Value Description 0 Odd parity is performed, which checks for an odd number of 1s. 1 Even parity generation and checking is performed during transmission and reception, which checks for an even number of 1s in data and parity bits. This bit has no effect when parity is disabled by the PEN bit.
1	PEN	RW	0	UART Parity Enable Value Description 0 Parity is disabled and no parity bit is added to the data frame. 1 Parity checking and generation is enabled.
0	BRK	RW	0	UART Send Break Value Description 0 Normal use. 1 A Low level is continually output on the UnTx signal, after completing transmission of the current character. For the proper execution of the break command, software must set this bit for at least two frames (character periods).

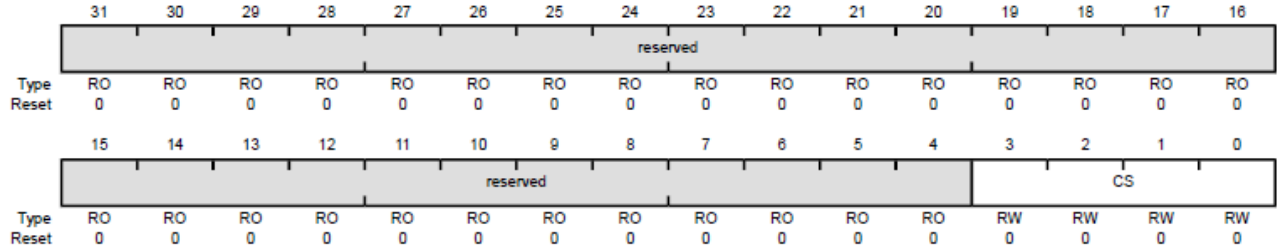
5. Configure the UART clock source by writing to the UARTCC register

939 / 1409

Register 18: UART Clock Configuration (UARTCC), offset 0xFC8

The **UARTCC** register controls the baud clock source for the UART module. For more information, see the section called "Communication Clock Sources" on page 222.

Note: If the PIOSC is used for the UART baud clock, the system clock frequency must be at least 9 MHz in Run mode.



3:0	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

// 6. Optionally, configure the µDMA channel (see “Micro Direct Memory Access (µDMA)” on page 585) and enable the DMA option(s) in the UARTDMACTL register ➔ **No need**
// 7. Enable the UART by setting the UARTEN bit in the UARTCTL register.

Register 8: UART Control (UARTCTL), offset 0x030

The **UARTCTL** register is the control register. All the bits are cleared on reset except for the Transmit Enable (TXE) and Receive Enable (RXE) bits, which are set.

To enable the UART module, the UARTEN bit must be set. If software requires a configuration change in the module, the UARTEN bit must be cleared before the configuration changes are written. If the UART is disabled during a transmit or receive operation, the current transaction is completed prior to the UART stopping.

Note: The **UARTCTL** register should not be changed while the UART is enabled or else the results are unpredictable. The following sequence is recommended for making changes to the **UARTCTL** register.

1. Disable the UART.
2. Wait for the end of transmission or reception of the current character.
3. Flush the transmit FIFO by clearing bit 4 (FEN) in the line control register (**UARTLCRH**).
4. Reprogram the control register.
5. Enable the UART.

	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

9	RXE	RW	1	UART Receive Enable

Value	Description
0	The receive section of the UART is disabled.
1	The receive section of the UART is enabled.

If the UART is disabled in the middle of a receive, it completes the current character before stopping.

Note: To enable reception, the `UARTEN` bit must also be set.

8	TXE	RW	1	UART Transmit Enable

Value	Description
0	The transmit section of the UART is disabled.
1	The transmit section of the UART is enabled.

If the UART is disabled in the middle of a transmission, it completes the current character before stopping.

Note: To enable transmission, the `UARTEN` bit must also be set.

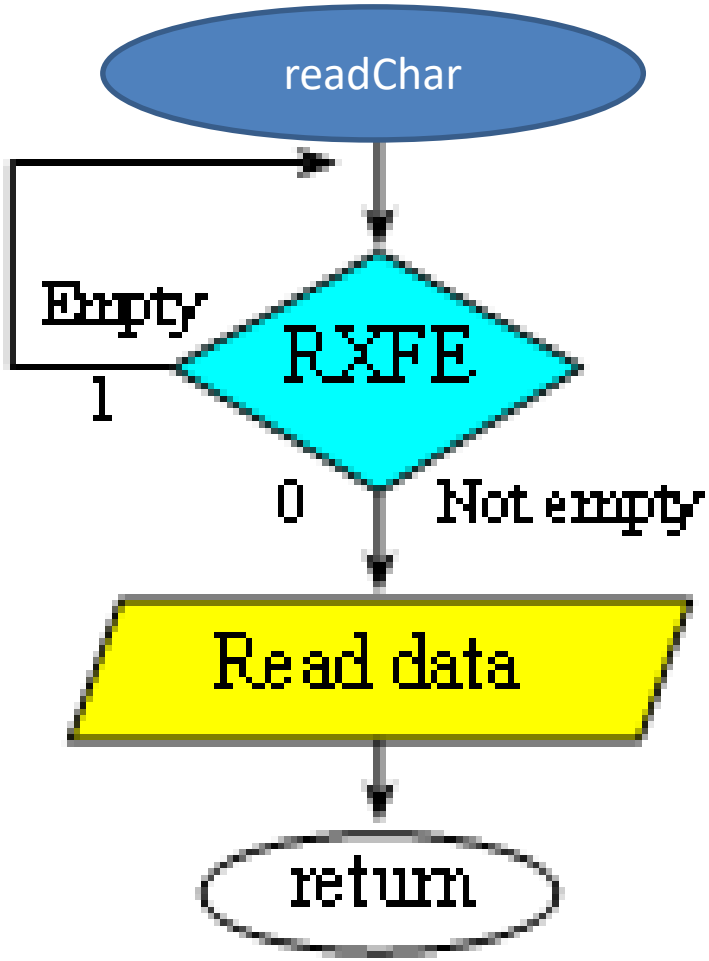
0	UARTEN	RW	0	UART Enable

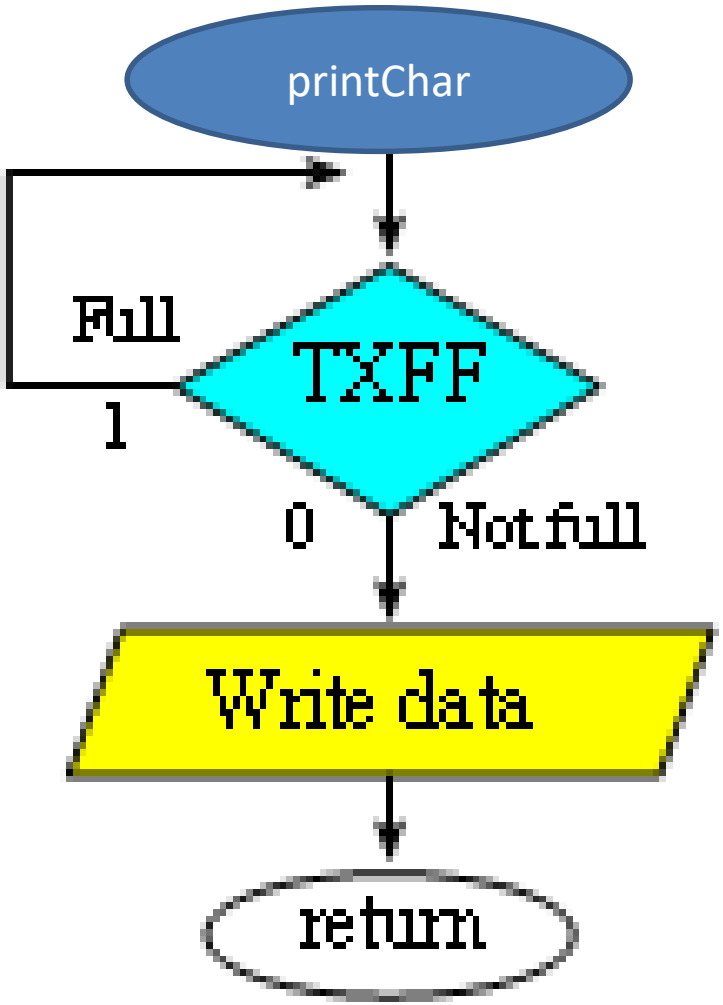
Value	Description
0	The UART is disabled.
1	The UART is enabled.

If the UART is disabled in the middle of transmission or reception, it completes the current character before stopping.

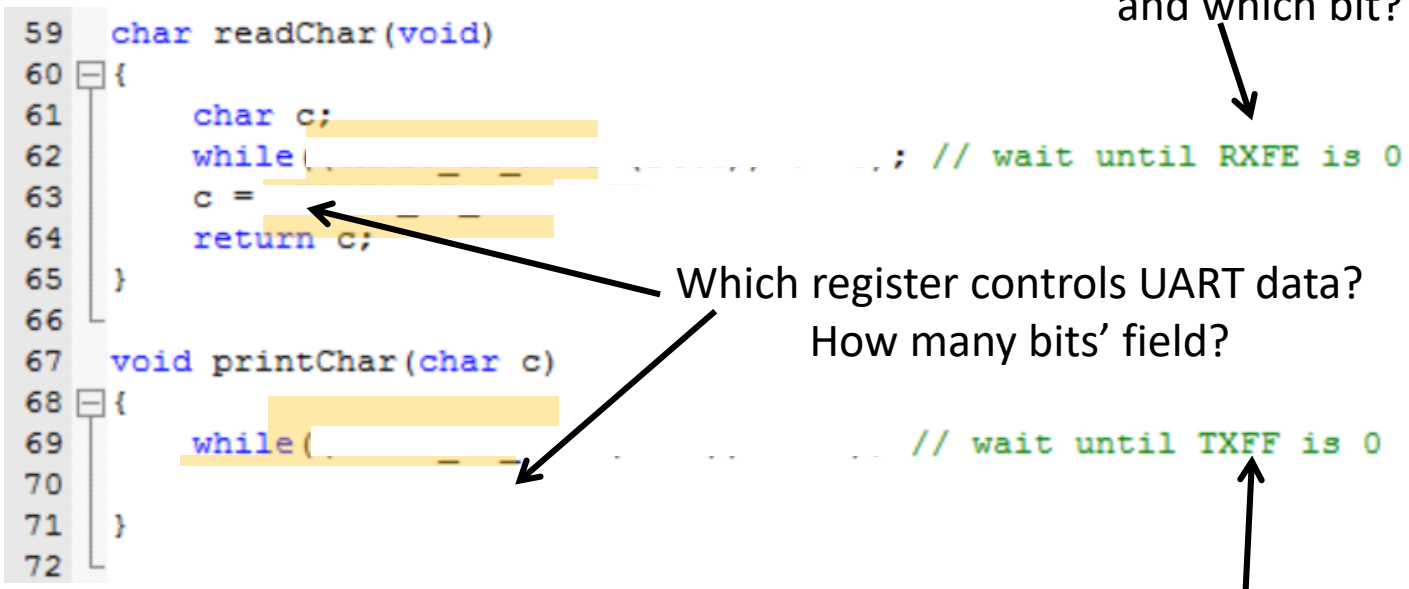
“UART Synchronization”

- Busy-wait operation





Busy-wait (flags)

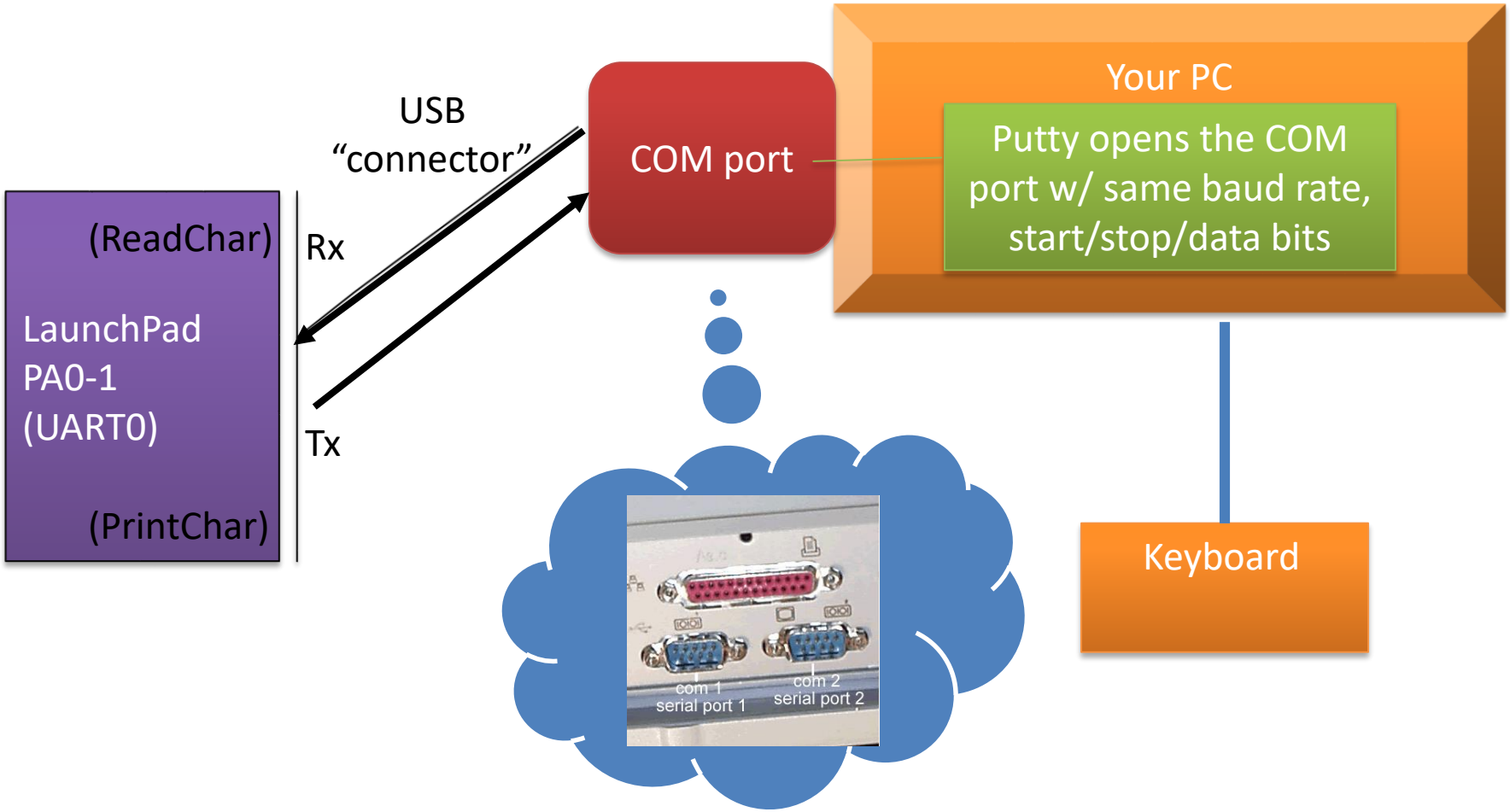


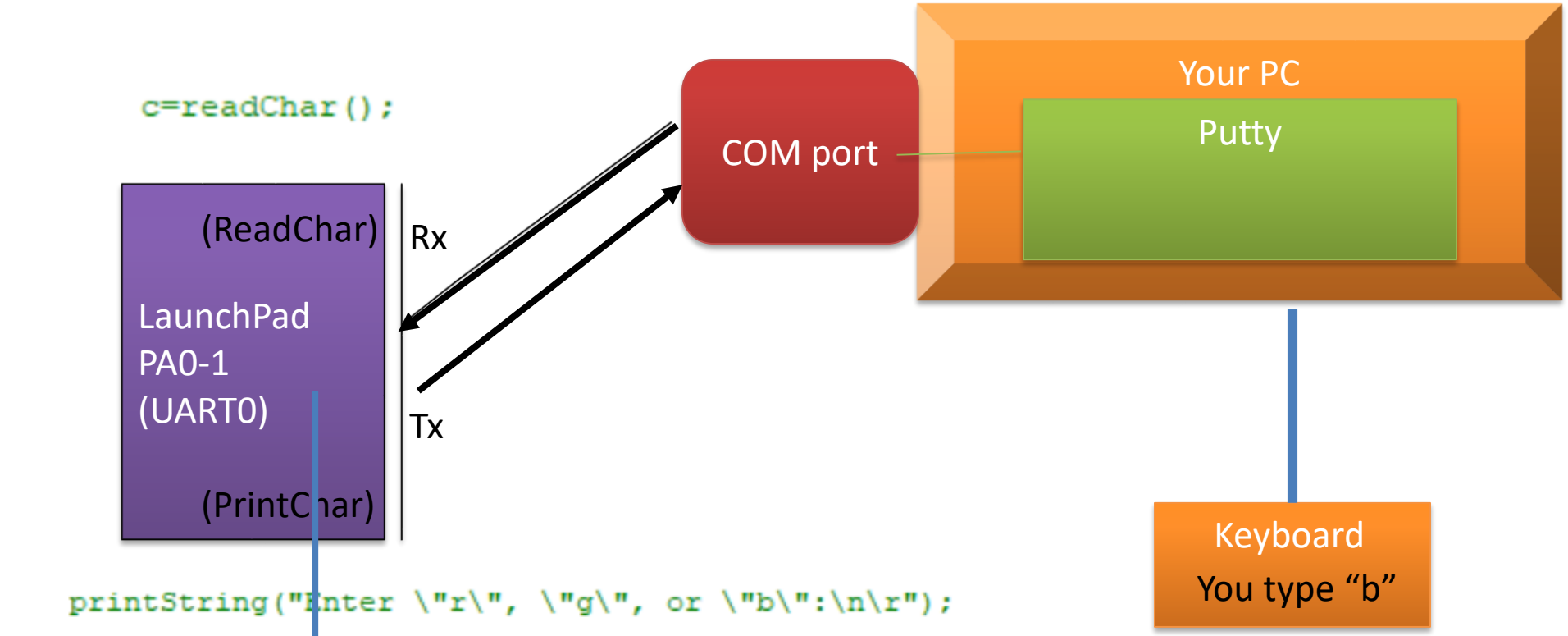
UART Flag (UART_FR)
UART DATA (UART_DR)

UART Demo

- Putty: serial communication software
  - http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
- Onboard LED control using UART0 (USB through PC)

UART lab story line

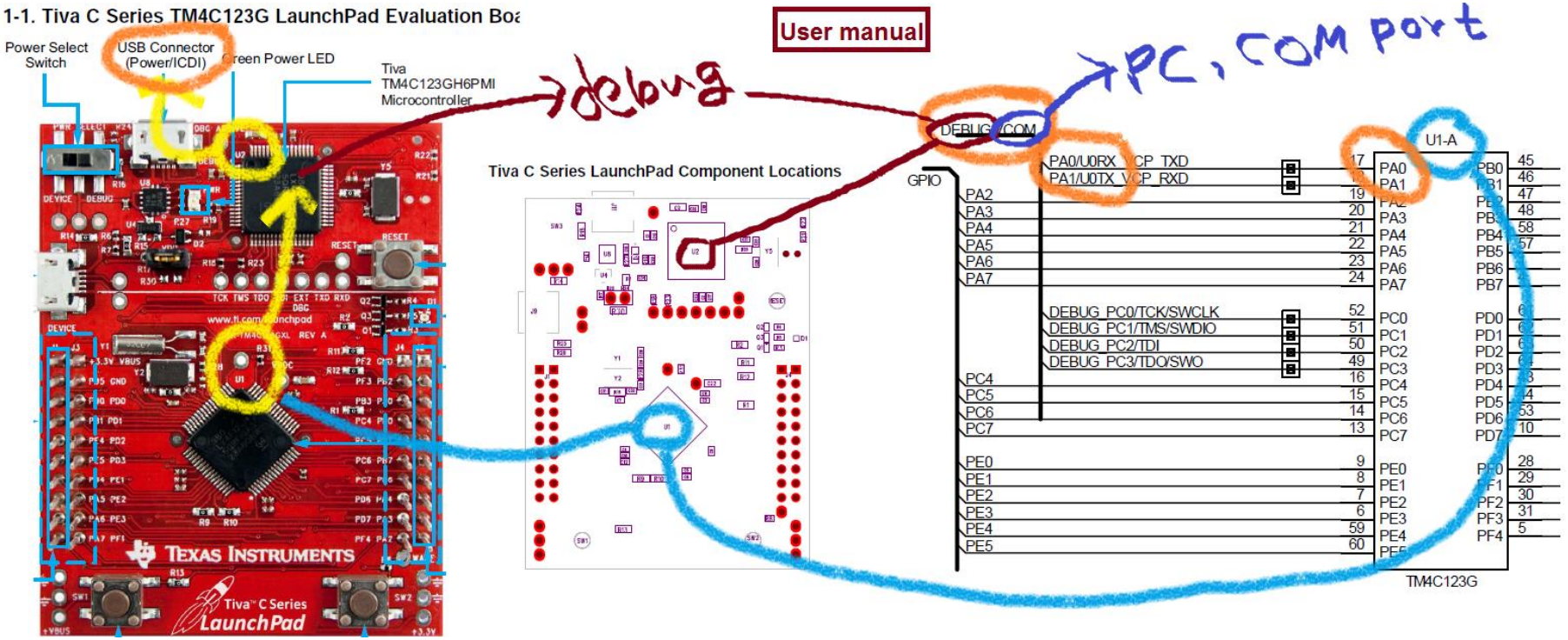


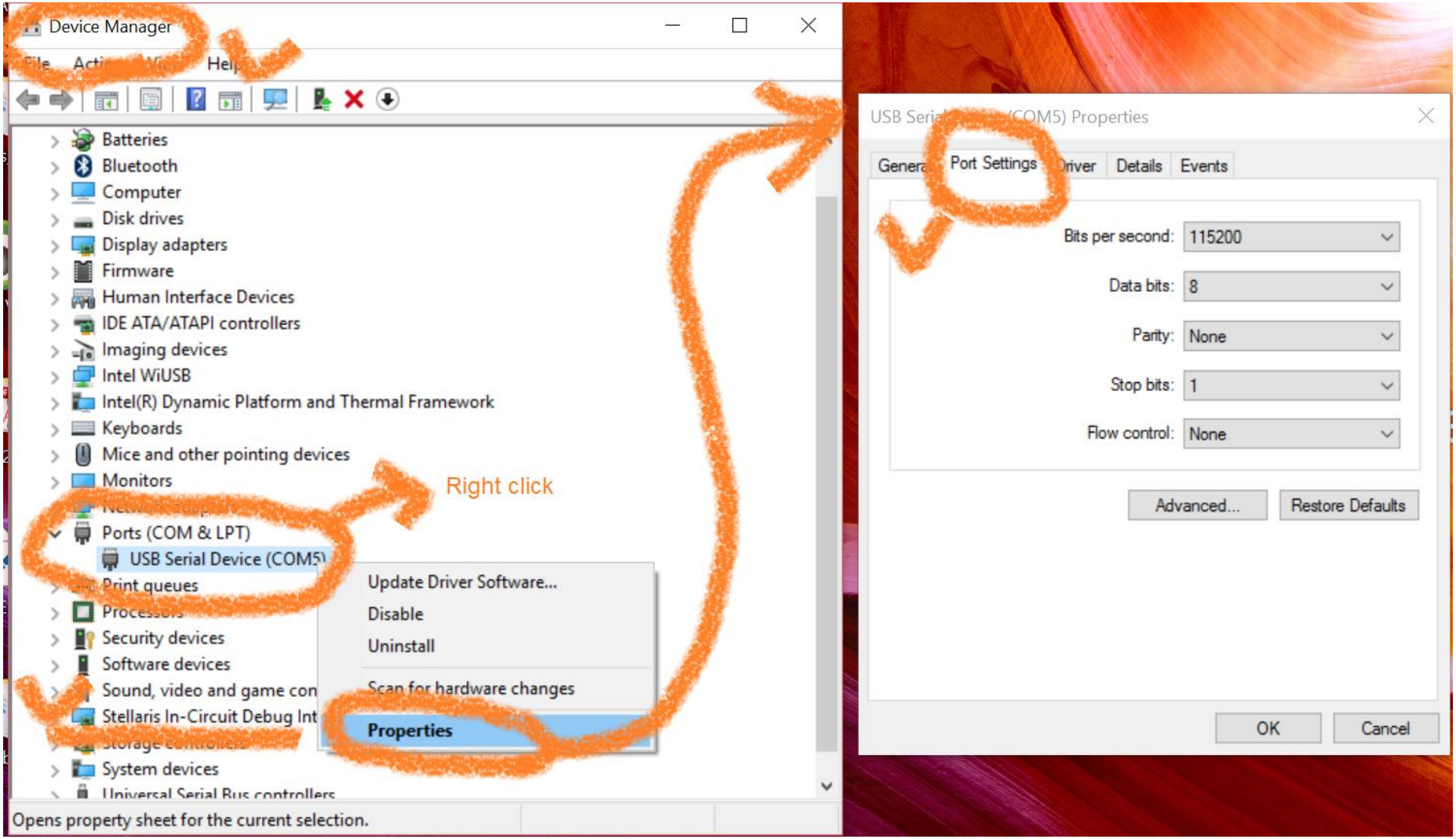


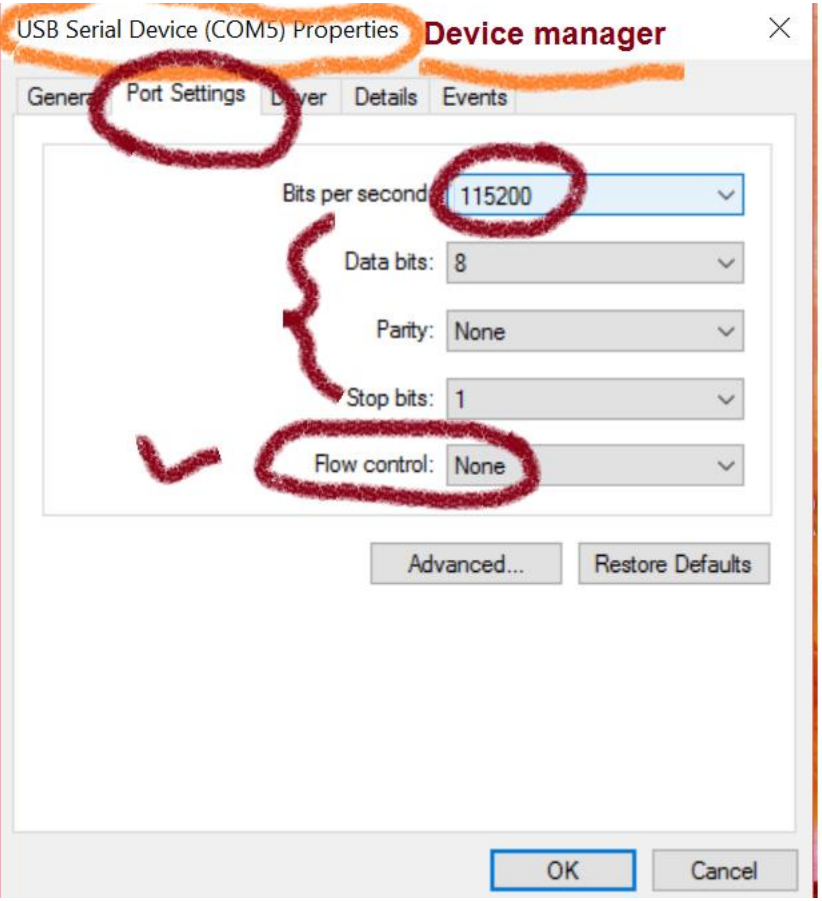

```
printString("Enter \"r\", \"g\", or \"b\":\n\r");
printChar(c);
```

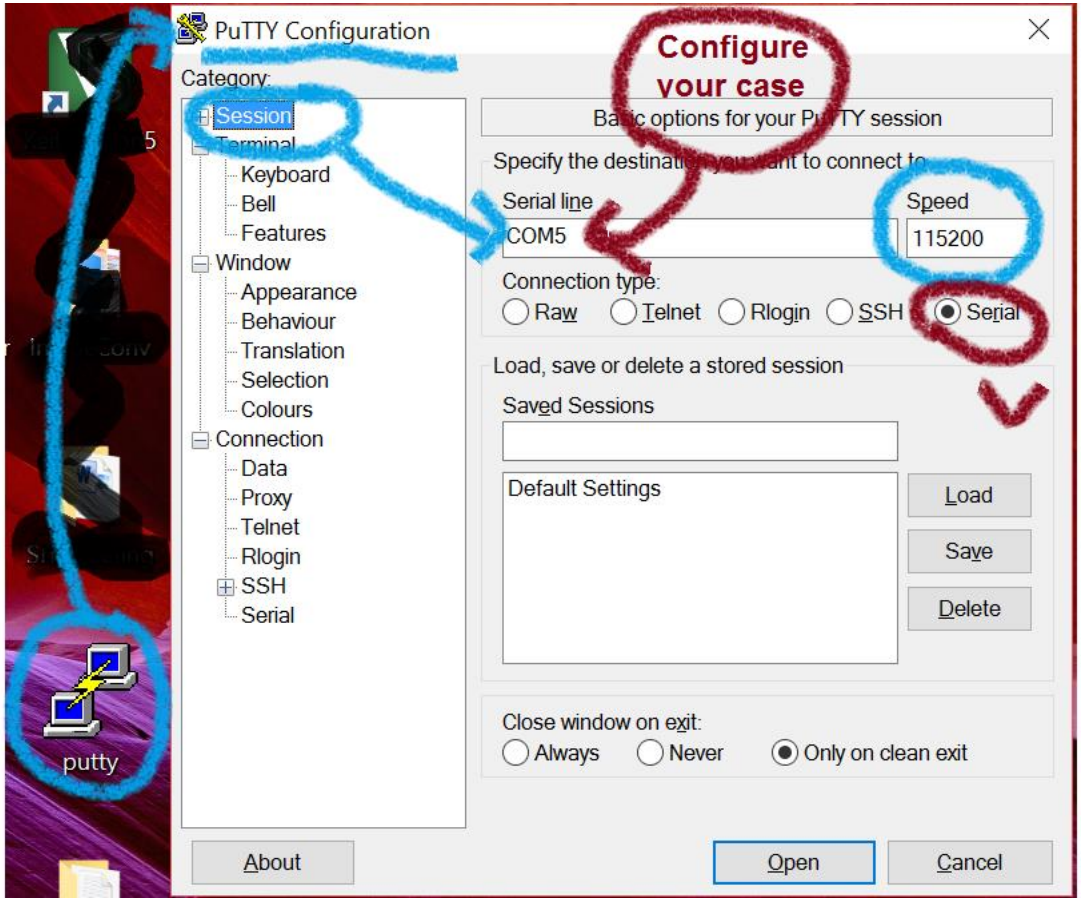
```
case 'b':
    GPIO_PORTF_DATA_R= (1<<2); //blue LED on
  break;
```

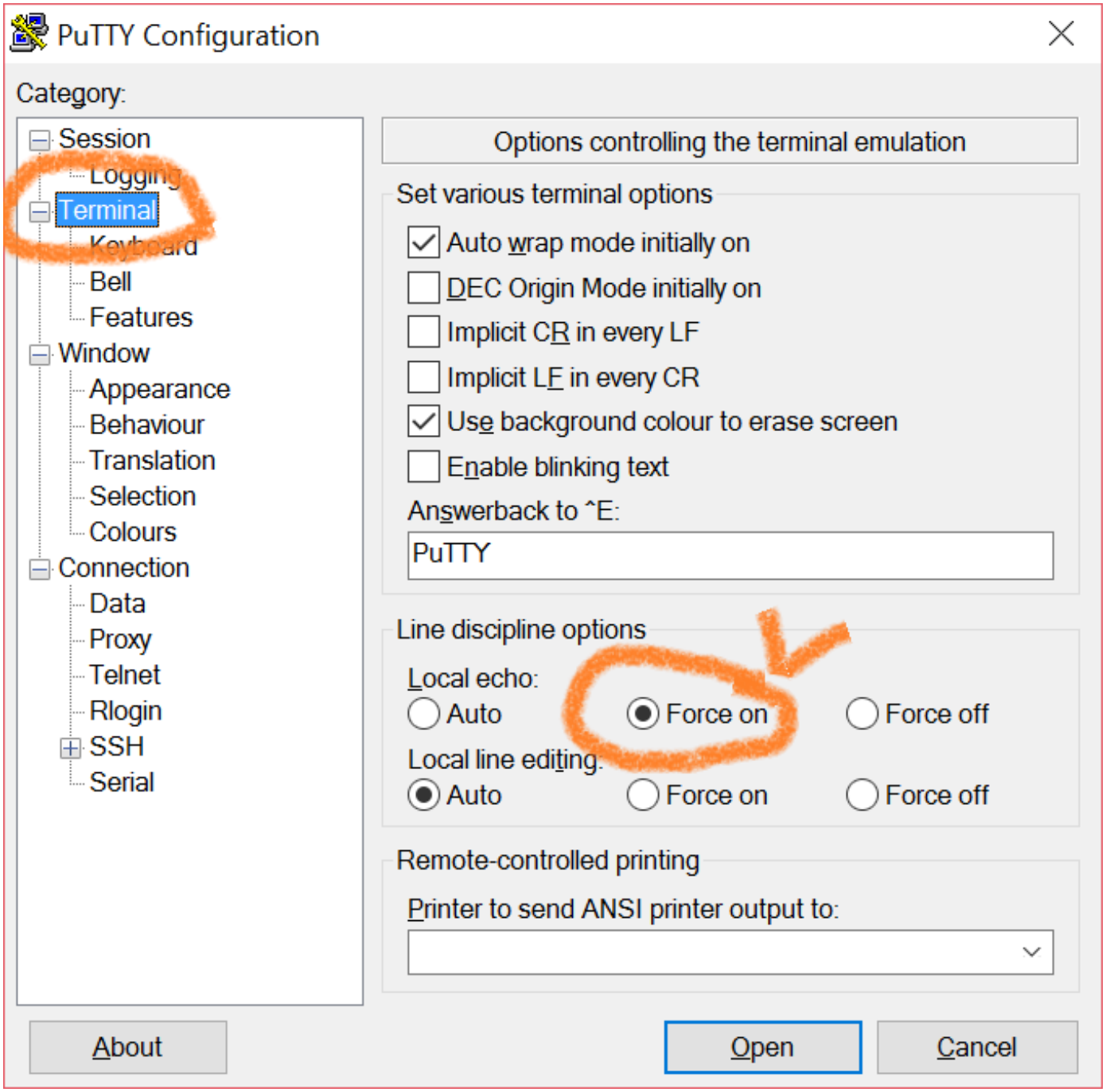
Figure 1-1. Tiva C Series TM4C123G LaunchPad Evaluation Boa

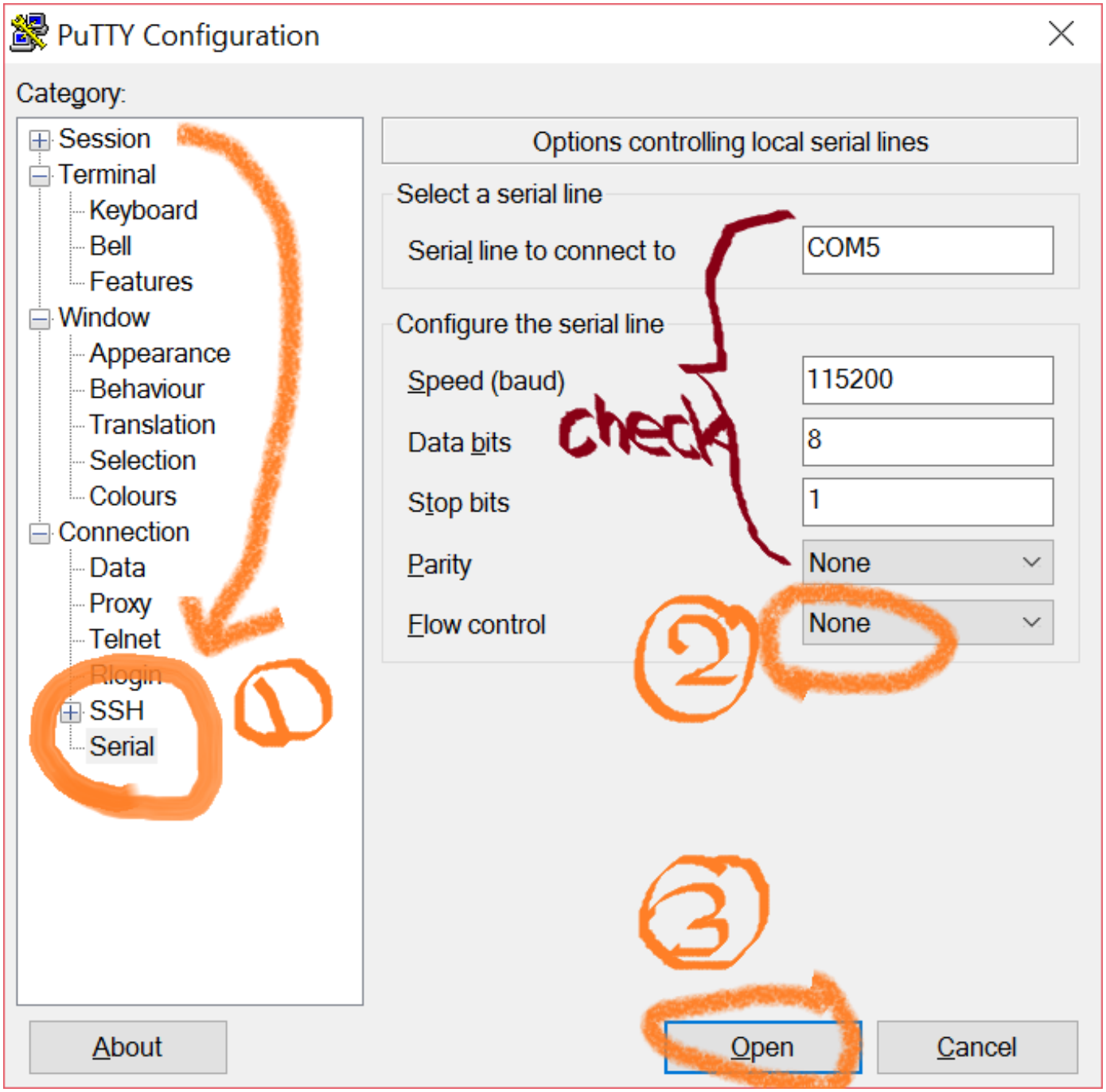


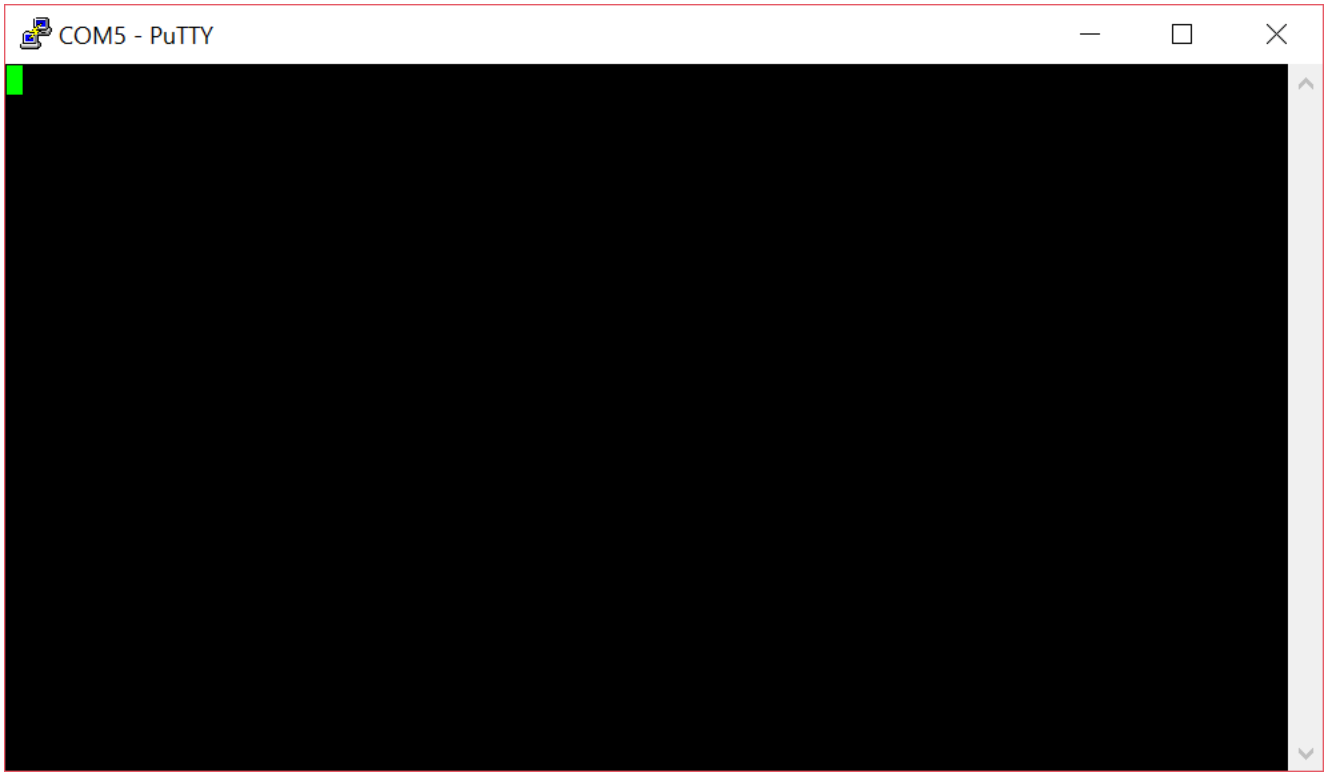

Device Manager
Batteries
Bluetooth
Computer
Disk drives
Display adapters
Firmware
Human Interface Devices
IDE ATA/ATAPI controllers
Imaging devices
Intel WiUSB
Intel(R) Dynamic Platform and Thermal Framework
Keyboards
Mice and other pointing devices
Monitors
Ports (COM & LPT)
USB Serial Device (COM5)
Print queues
Security devices
Software devices
Stellaris In-Circuit Debug Int
System devices
Right click
Update Driver Software...
Disable
Uninstall
Scan for hardware changes
Properties
Opens property sheet for the current selection.
Port Settings
Details
Events
Bits per second: 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None
Advanced...
Restore Defaults
OK
Cancel

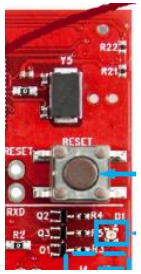

USB Serial Device (COM5) Properties
Device manager
Port Settings
Details
Events
Bits per second: 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None
Advanced...
Restore Defaults
OK
Cancel


PuTTY Configuration
Configure your case
Category:
Session
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Serial line
Speed
COM5
115200
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel
putty

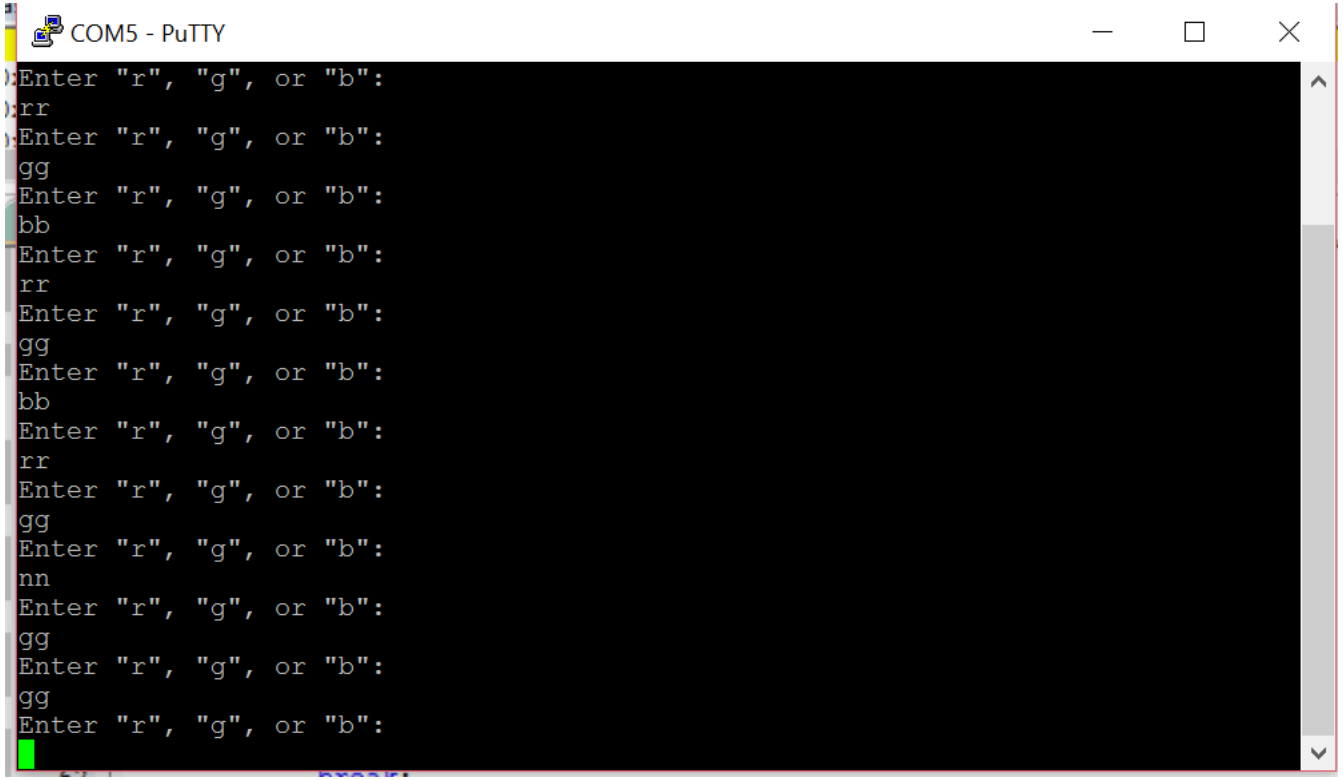

PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Options controlling the terminal emulation
Set various terminal options
Auto wrap mode initially on
DEC Origin Mode initially on
Implicit CR in every LF
Implicit LF in every CR
Use background colour to erase screen
Enable blinking text
Answerback to ^E:
PuTTY
Line discipline options
Local echo:
Auto
Force on
Force off
Local line editing:
Auto
Force on
Force off
Remote-controlled printing
Printer to send ANSI printer output to:
About
Open
Cancel


PuTTY Configuration
Category:
Session
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Options controlling local serial lines
Select a serial line
Serial line to connect to
COM5
Configure the serial line
Speed (baud)
115200
Data bits
8
Stop bits
1
Parity
None
Flow control
None
check
1
2
3
About
Open
Cancel





Press Reset button on your board


COM5 - PuTTY
Enter "r", "g", or "b":
rr
Enter "r", "g", or "b":
gg
Enter "r", "g", or "b":
bb
Enter "r", "g", or "b":
rr
Enter "r", "g", or "b":
gg
Enter "r", "g", or "b":
bb
Enter "r", "g", or "b":
rr
Enter "r", "g", or "b":
gg
Enter "r", "g", or "b":
nn
Enter "r", "g", or "b":
gg
Enter "r", "g", or "b":
gg
Enter "r", "g", or "b":

Reading

Vol.1	Vol.2
Ch.8 (8.1, 8.2, 8.3)	Ch.4 (4.9) Ch.7 (7.1, 7.5, 7.6, 7.7…)