EEL 4742 – Embedded Systems
Module 6 – UART

Hadi M Kamali

Department of Electrical and Computer Engineering (**ECE**)
University of Central Florida

Office Location/phone: HEC435 – (407) 823-0764
webpage: https://www.ece.ucf.edu/~kamali/
e-mail: kamali@ucf.edu
HAVEN Research Group
https://haven.ece.ucf.edu/

HARDWARE VERIFICATION ENGINEERING ASSURANCE

UCF UNIVERSITY OF CENTRAL FLORIDA

What is UART

- Universal Asynchronous Receiver and Transmitter (UART)
  - It is a serial communication protocol
  - Mostly between the microcontroller/microprocessors and external environment
  - Data is usually on transfer bit-by-bit
    - It is as opposed to parallel communication, where the data is transferred in bytes!
- UART is ASYNCHRONOUS
  - There is no shared clock reference between two devices
  - Each side of communication has its own clock source.

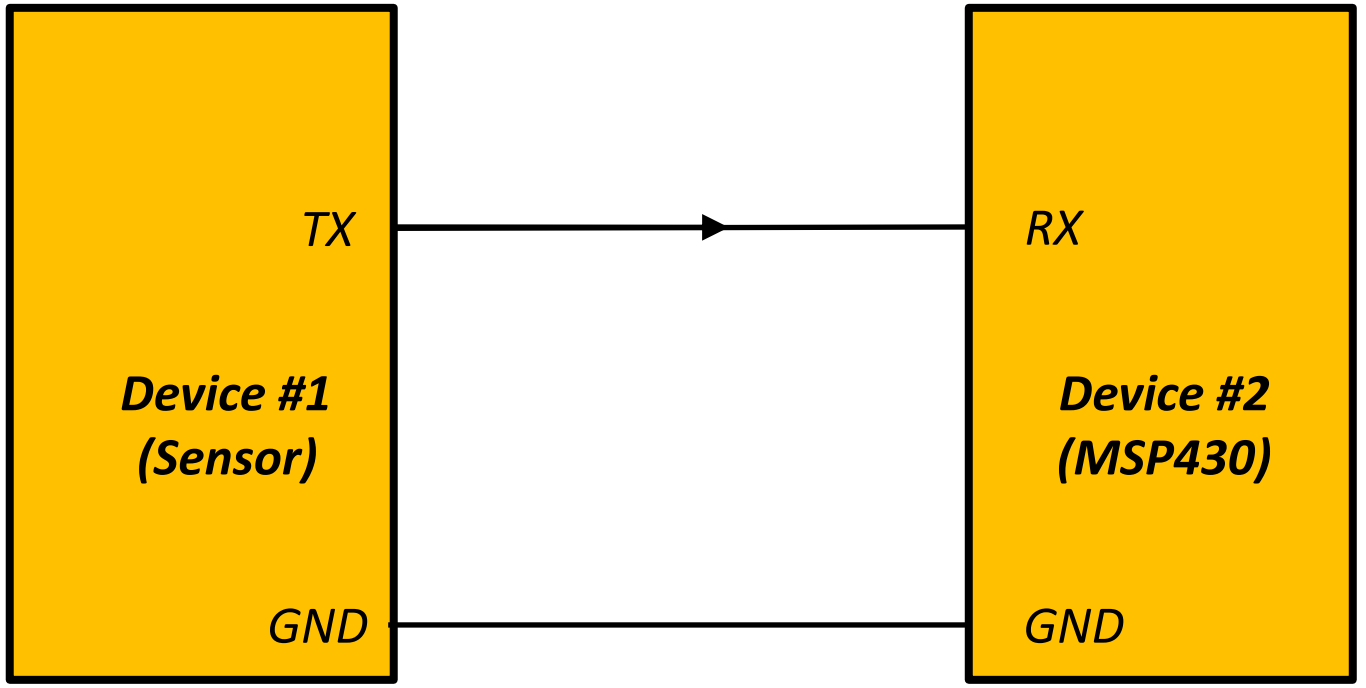
simplifies the wiring but requires that both ends agree on a communication speed (baud rate).

(bits per second)

UART Operation Modes

- Simplex → data flow in one direction

One device transmits!
One device receives!



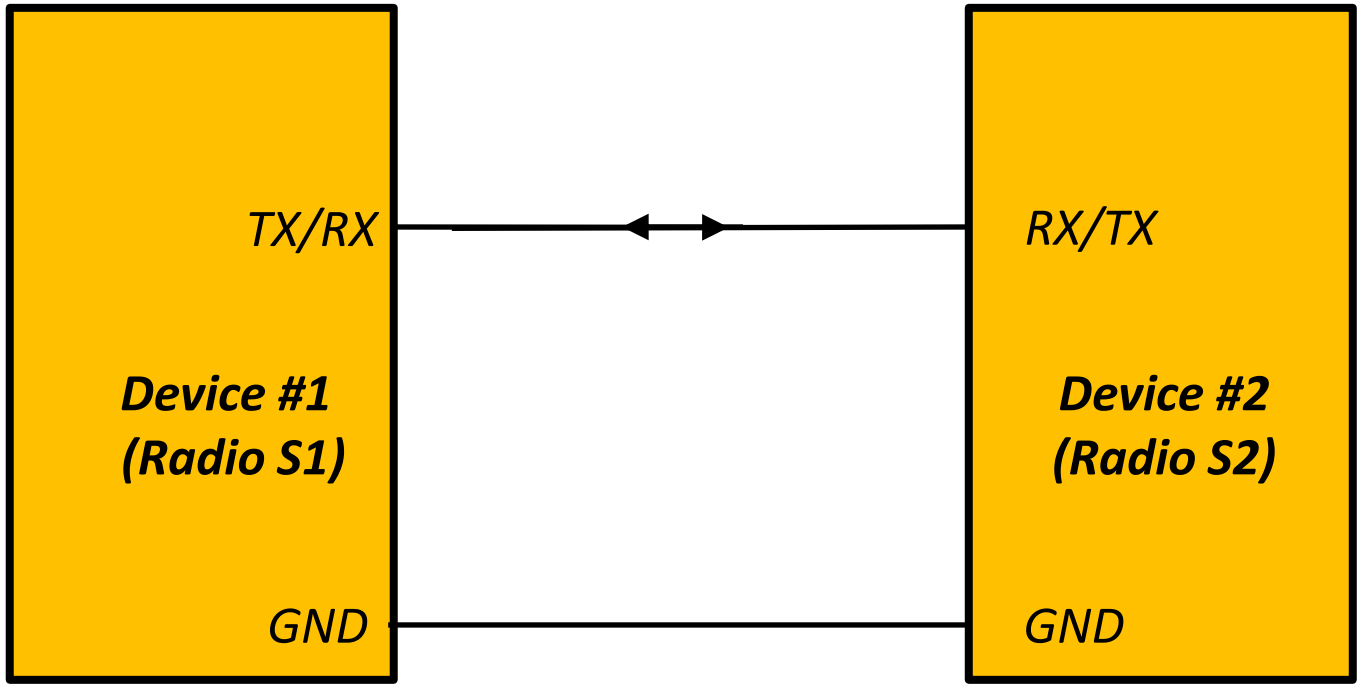
Example: *A sensor transmitting data to a microcontroller, where the sensor can only send data and never receive it.*

In such a case, there is only one wire required (in addition to the ground)

UART Operation Modes

- **Half-Duplex → data flow in both direction (not at the same time)**

The system must alternate between transmitting and receiving.



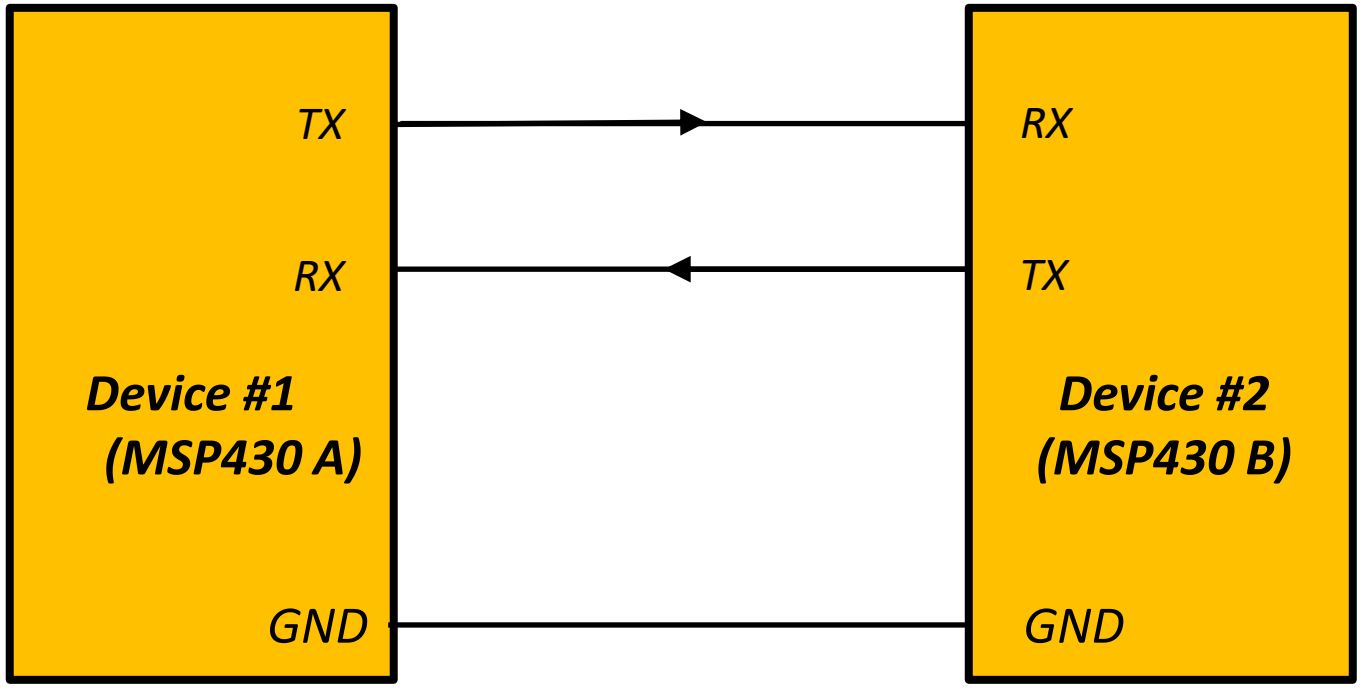
Example: *A two-way radio communication where only one person can speak at a time.*

In such a case, there is only one wire required (in addition to the ground)

UART Operation Modes

- **Full-Duplex → data flow in both direction (at the same time)**

The system transmits and receives at the same time through different wirings.



Example: *Communication between two microcontrollers, where both need to send and receive data without waiting for the other to finish.*

It requires two wires dedicated for one direction of communication each (in addition to the ground).

UART Data Transmission

- In a communication using UART
  - The transmitter controls the voltage level on the data wire.
  - The receiver only senses the voltage level.

The transmitter in a UART system is responsible for sending data by converting digital signals (logic 1s and 0s) into corresponding voltage levels on the transmission line.

The receiver in UART needs to detect the voltage on the transmission line and convert it back into digital data (1s and 0s). The receiver's role is to sense these voltage levels accurately and decide whether the incoming signal represents a logic 1 or a logic 0.

- Two common voltage levels

More for longer transmission distances (better noise immunity)

Transistor-Transistor Logic (TTL)

logic 1**: represented by a voltage of **+3.3V** or **+5V

logic 0**: represented by a voltage of **0V

RS-232

logic 1**: represented by a voltage of **+3V** or **+15V

logic 0**: represented by a voltage of **-3V** or **-15V

UART Data Transmission

- In a communication using UART
  - The transmitter controls the voltage level on the data wire.
  - The receiver only senses the voltage level.

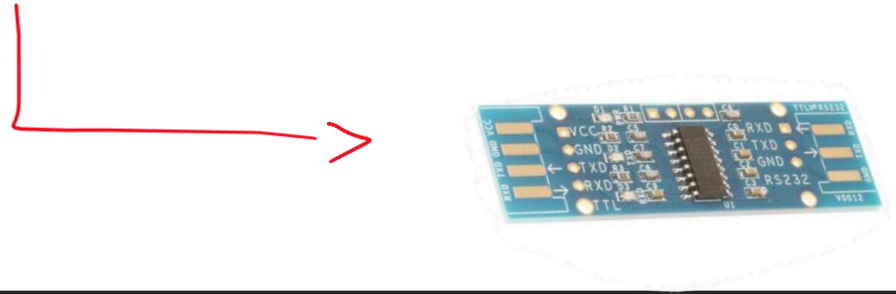
TTL-to-TTL Communication

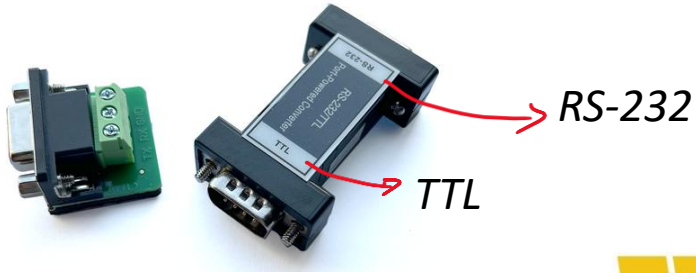
- *The transmitter will send 3.3V/5V and 0V for logic 1 and 0, and the receiver will correctly interpret these signals.*

RS-232-to-RS-232 Communication

- *The transmitter sends positive and negative voltages, which the receiver will correctly detect as logic 0 and 1.*

*If there is a mismatch between voltage levels (e.g., if a TTL device is connected to an RS-232 device directly), a **level shifter** or **converter** is needed to ensure the correct voltage translation between the two standards.*



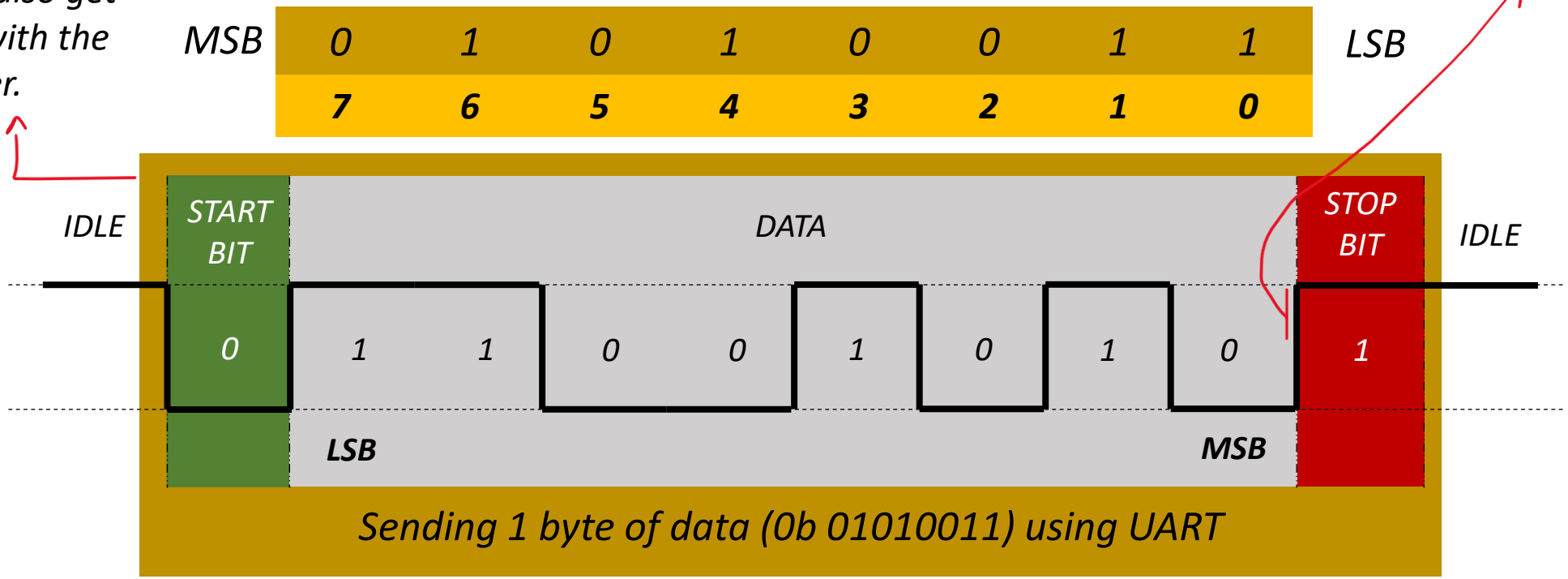


UART Data Transmission

- Sending data bit-by-bit
  - It is typical to send the least significant bit first.

*Once all the data bits have been sent, the optional **parity bit** (if enabled) is transmitted for error-checking purposes.*

Receivers also get the data with the same order.



UART Data Transmission

- **Distinguishing between start bit, data bits, and stop bit(s)**

Start bit detection

Idle State: *When no data is being transmitted, the UART line remains in an* ***idle state (logic 1)***

Start Bit as a Low Signal: *A low voltage level (logic 0). The start bit is the time the line transitions from idle (high) to low, making it easy to detect.*

The UART receiver constantly monitors the line. When it detects the transition from ***high (idle)*** *to* ***low (start bit)****, it knows that data transmission is starting.*

Data bits

Baud Rate Synchronization: *Once the start bit is detected, the receiver uses the agreed-upon* ***baud rate*** *(bits per second) to know when to sample the incoming bits.*

*The UART protocol defines how many data bits will be transmitted (****usually 8 bits****, but it can be between 5 to 9 bits). The receiver knows to expect this* ***specific number of bits****.*

Stop bit(s)

Stop Bit as a High Signal: *After the last data bit is transmitted, the* ***stop bit*** *is sent. The stop bit is always a* ***high voltage level (logic 1)****, which corresponds to the* ***idle state*** *of the line.*

The stop bit signals the end of the data frame. The line remains in the ***high state (idle)*** *after the stop bit until another frame begins.*

UART Configuration

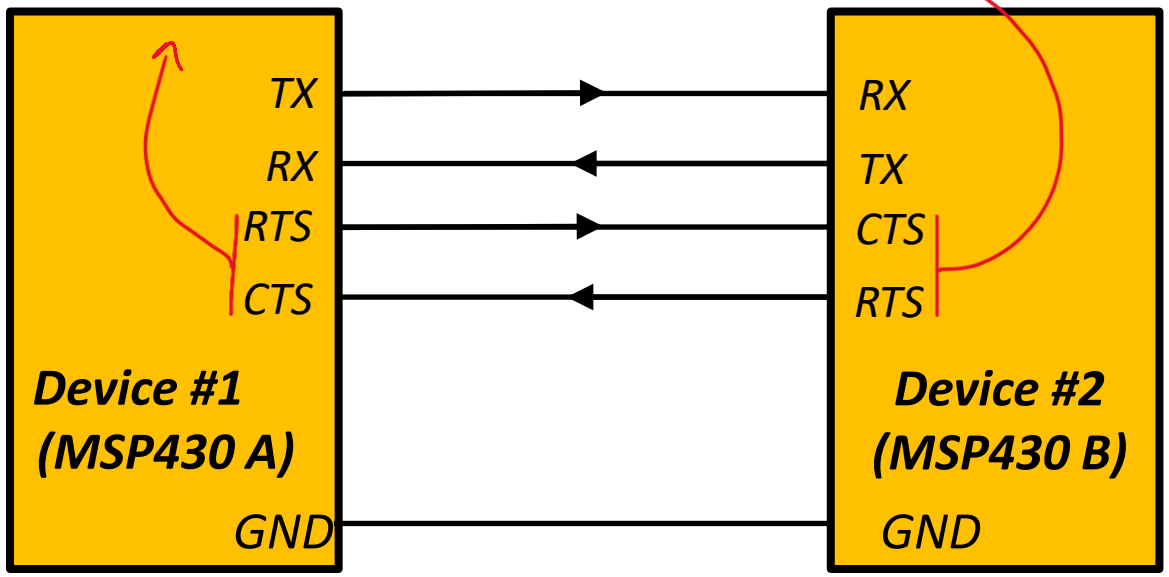
- Important Parameters for UART Communication

Parameter	Description	Options	Typical configuration
Baud Rate	Data rate / transmission speed (bits per second)	1200, 2400, 4800, 9600, 19200, 38400 and so on.	9600 baud
Data size	Number of bits in the data	5,6,7,8	8 bits
Bit ordering	Order of the data to be sent	LSB first or MSB first	LSB first
Parity bit	Parity bit for error detection	Odd, even or none	None
Stop bits	A way to signal the end of data transmission	1 bit or 2 bits	1 bit
Flow control	Optional way to control the flow of data between the devices	Hardware flow control, Software flow control, None	None

Flow Control in UART

- Generally implemented in two ways
  - Hardware
  - Software

Ready-to-Send (RTS) and Clear-to-Send (CTS) are two additional optional lines, that can be used to enable hardware flow control.



RTS

- ***RTS*** *is used by the transmitter.*
- *It is asserted (sets low) when it is ready to send.*
- *It is deasserted (sets high) when it is not ready to send.*

CTS

- ***CTS*** *is used by the receiver.*
- *It is asserted (sets low) when it is ready to receive.*
- *It is deasserted (sets high) when it is not ready to receive.*

Flow Control in UART

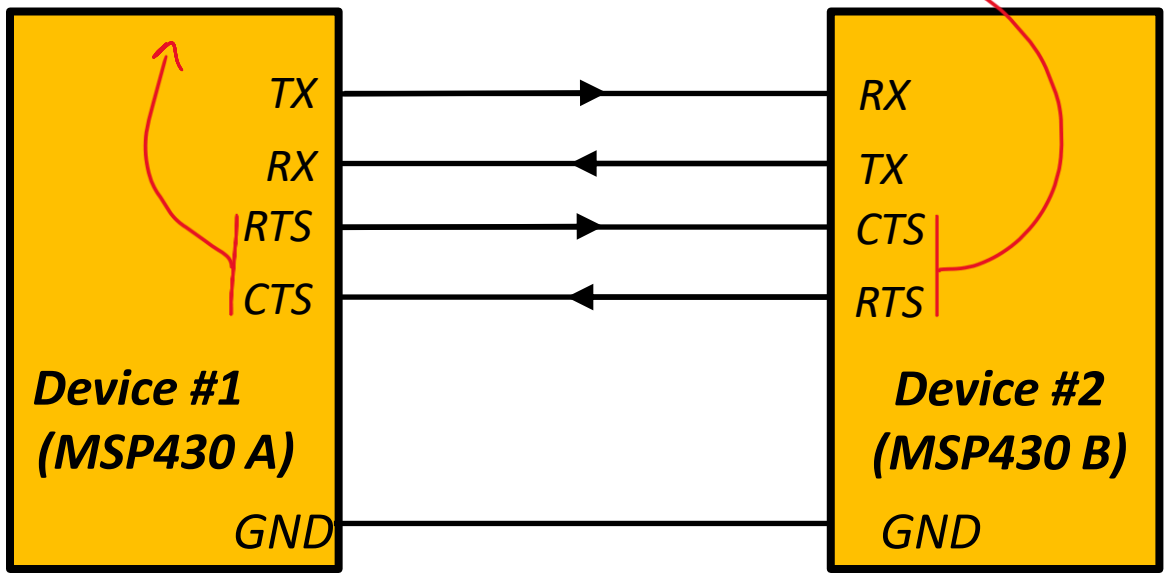
- Generally implemented in two ways
  - Hardware
  - Software

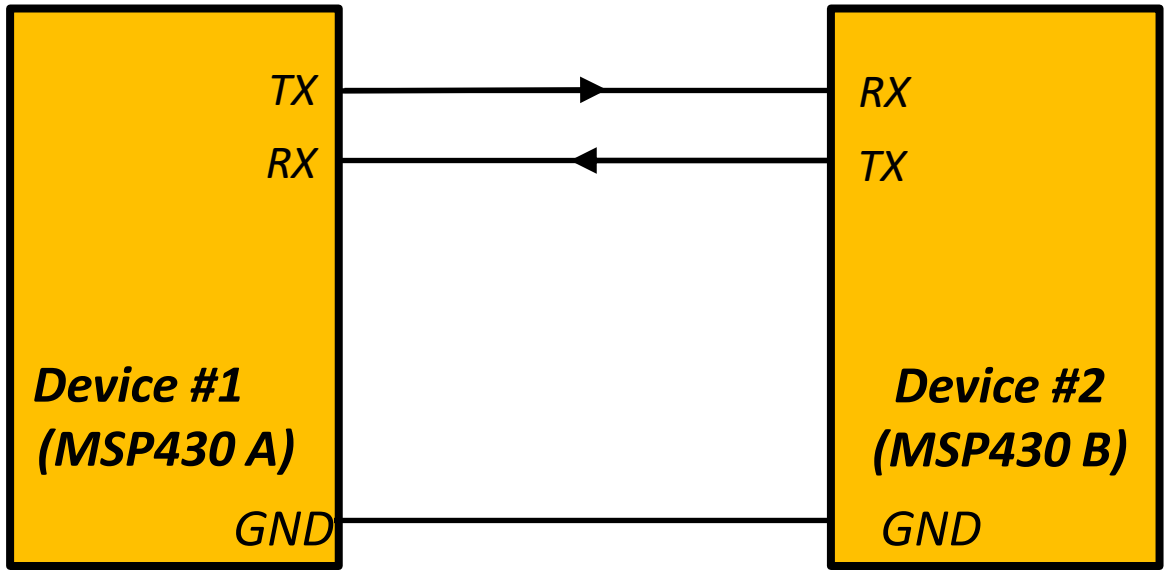
ASCII value 0x13

ASCII value 0x11

Ready-to-Send (RTS) and Clear-to-Send (CTS) are two additional optional lines, that can be used to enable hardware flow control.

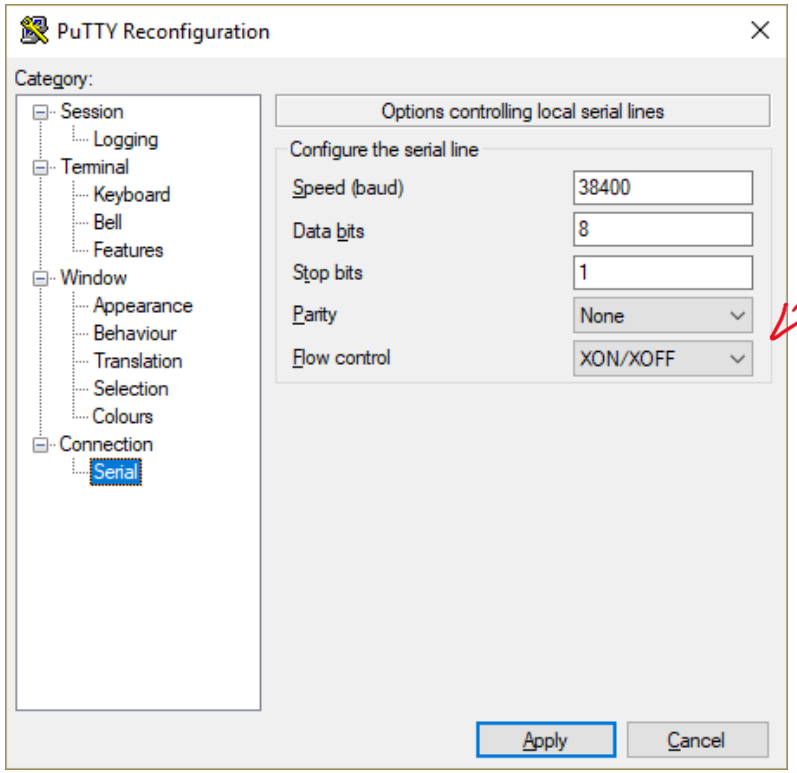
Software controls the flow (using XOFF/XON) signals. XON/XOFF are special channels sent on TX/RX.





Flow Control in UART

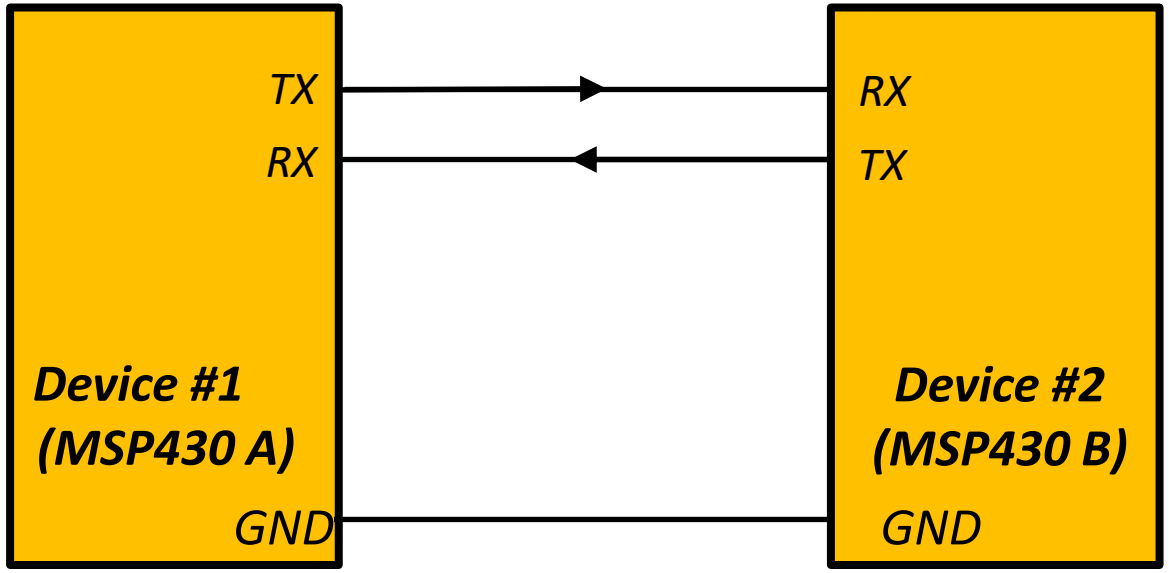
- Generally implemented in two ways
  - Hardware
  - Software



ASCII value 0x13

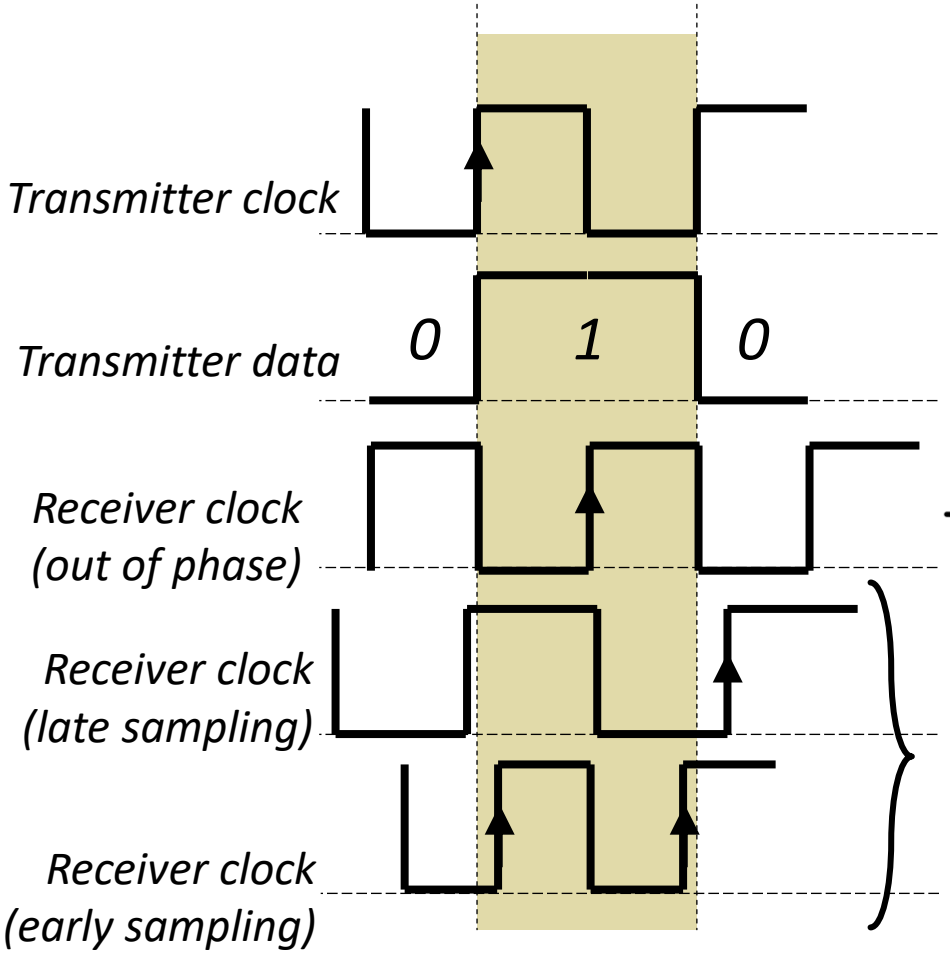
ASCII value 0x11

Software controls the flow (using XOFF/XON) signals. XON/XOFF are special channels sent on TX/RX.



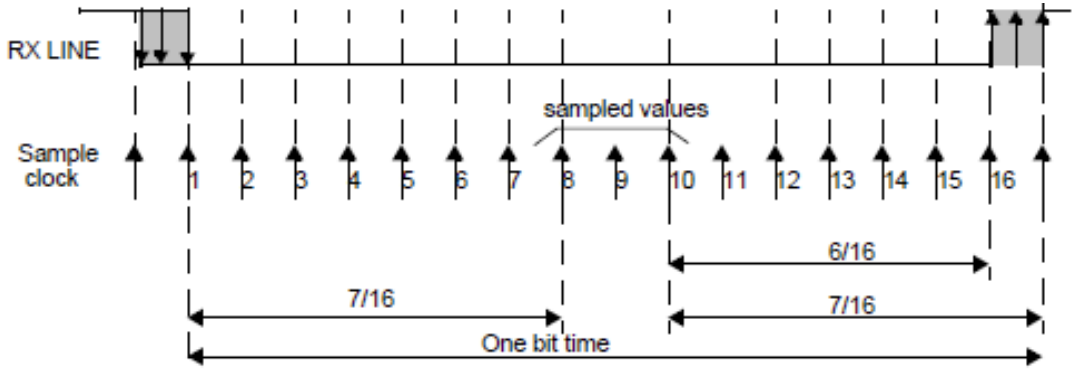
Oversampling in UART

- **UART is an ASYNCHRONOUS protocol. No shared clock.**



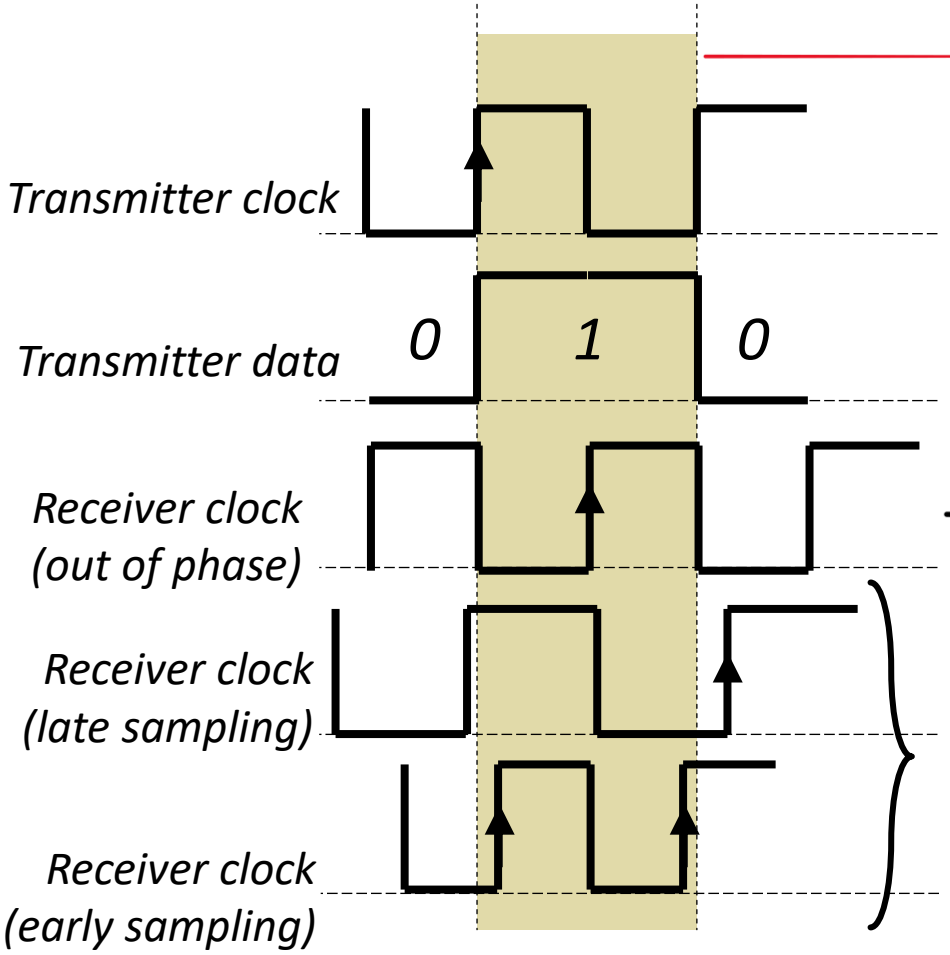
Best case scenario (Same frequency)

At high frequencies, small deviations in the sampling signal frequency could corrupt the data.



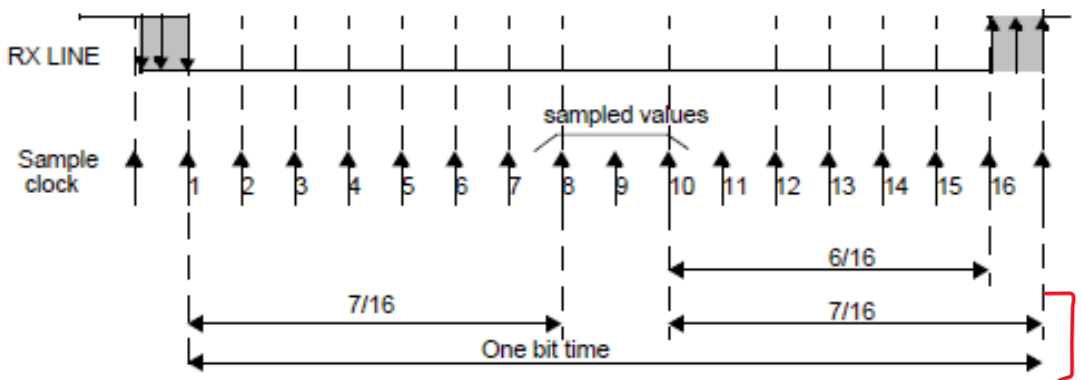
Oversampling in UART

- **UART is an ASYNCHRONOUS protocol. No shared clock.**



Best case scenario
(Same frequency)

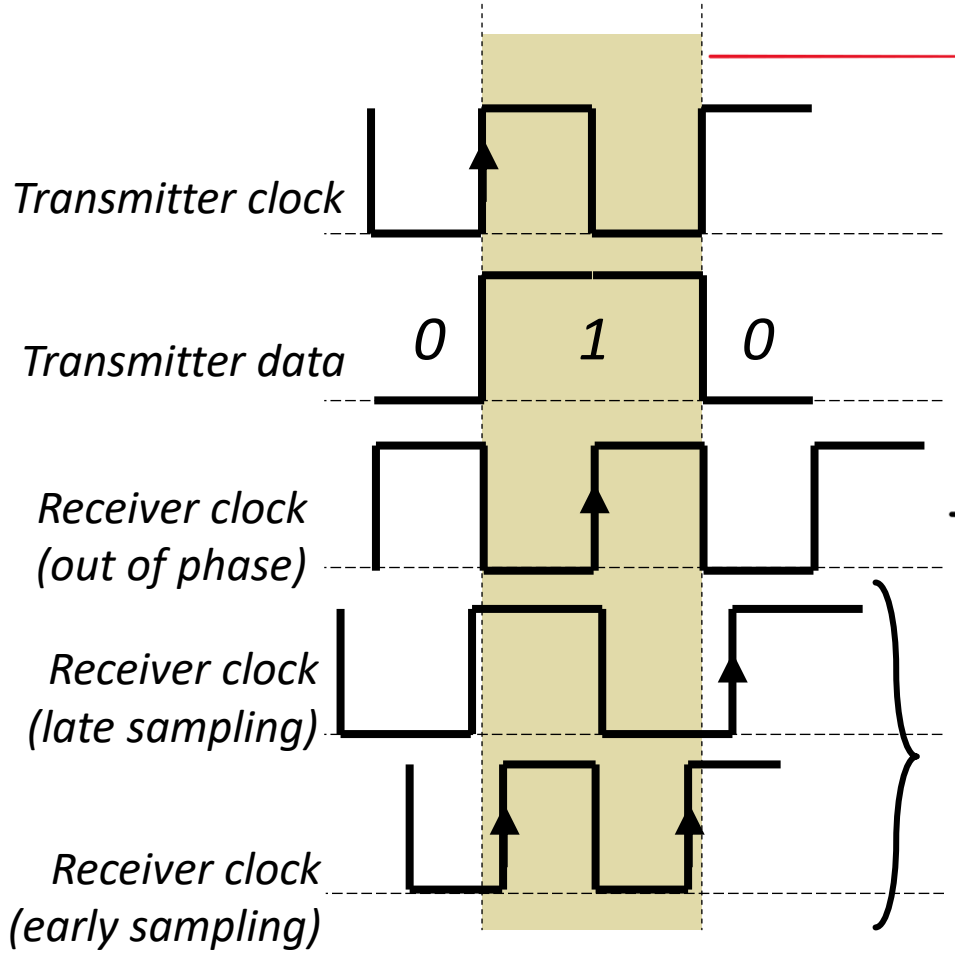
At high frequencies, small deviations in the sampling signal frequency could corrupt the data.



Oversampling 16x

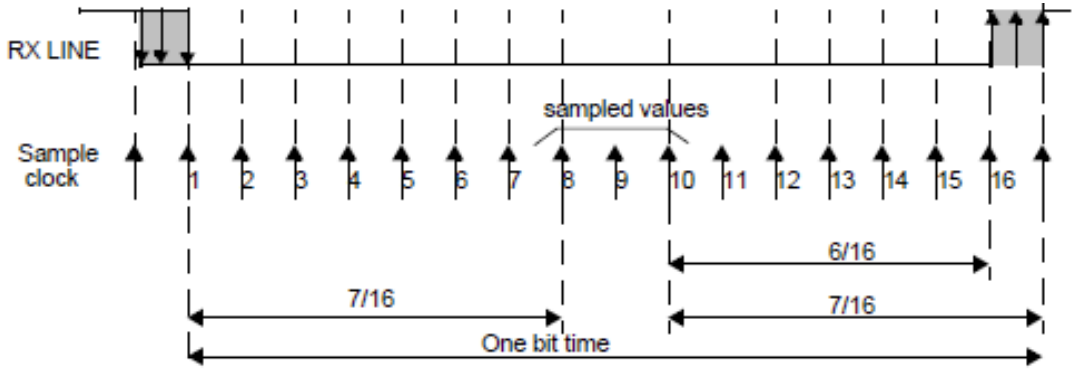
Oversampling in UART

- **UART is an ASYNCHRONOUS protocol. No shared clock.**



Best case scenario
(Same frequency)

At high frequencies, small deviations in the sampling signal frequency could corrupt the data.



Majority

*a **majority voting** technique to determine the bit's value*

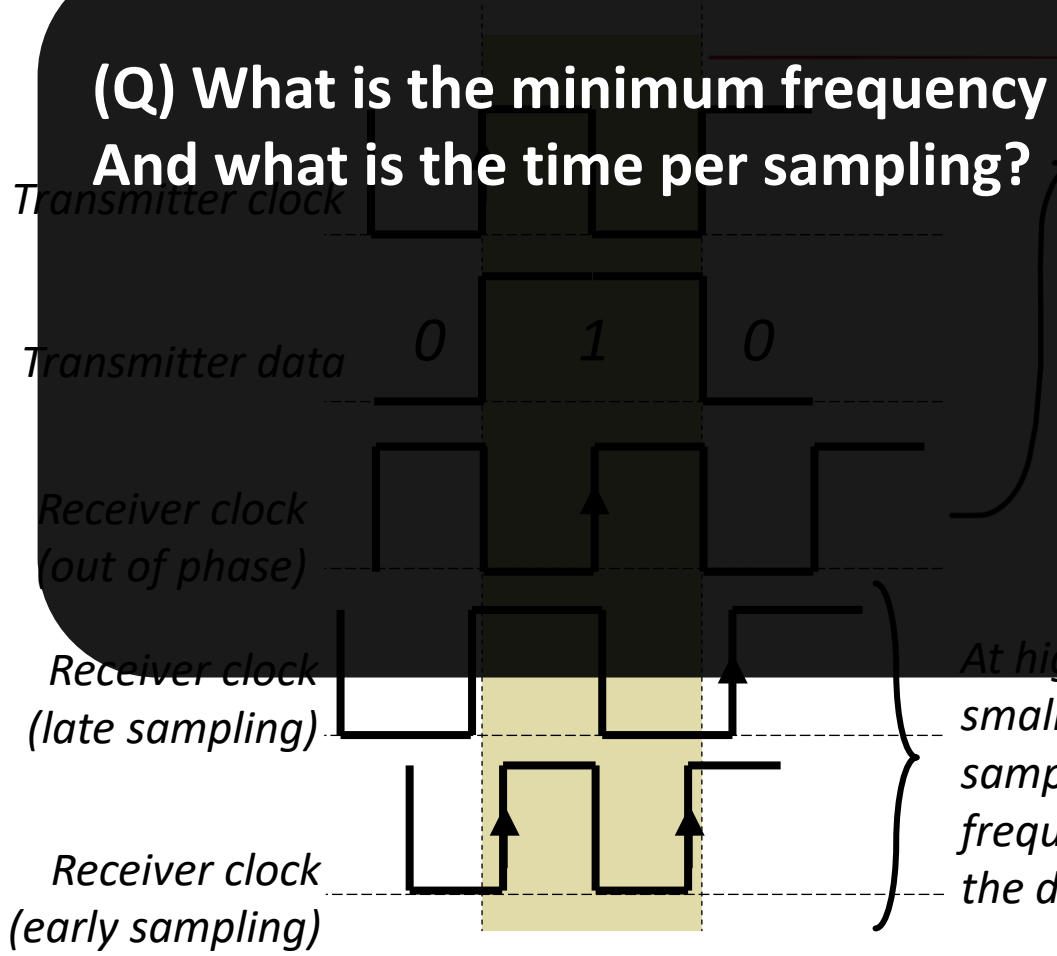
Median

*The **middle samples** (around the 8th or 9th sample in 16x oversampling)*

Oversampling in UART

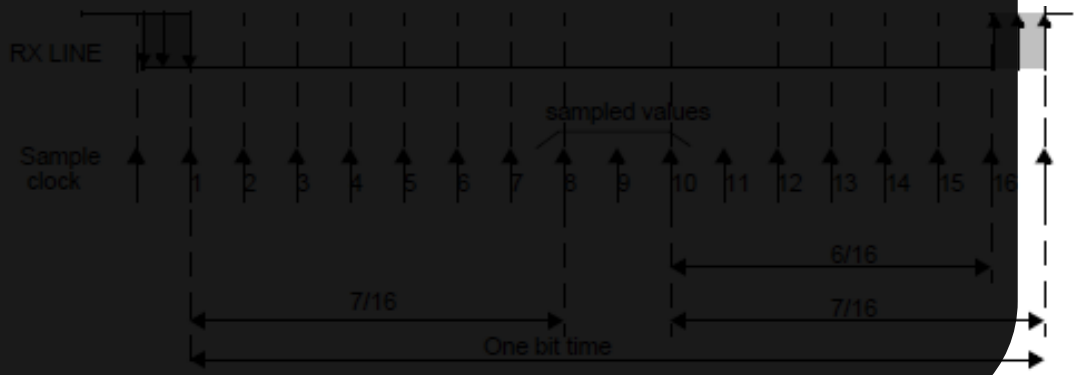
- UART is an ASYNCHRONOUS protocol. No shared clock.

(Q) What is the minimum frequency for a system at 9600 baud with 16x Oversampling? And what is the time per sampling?



Best case scenario (Same frequency)

At high frequencies, small deviations in the sampling signal frequency could corrupt the data.



Majority

*a **majority voting** technique to determine the bit's value*

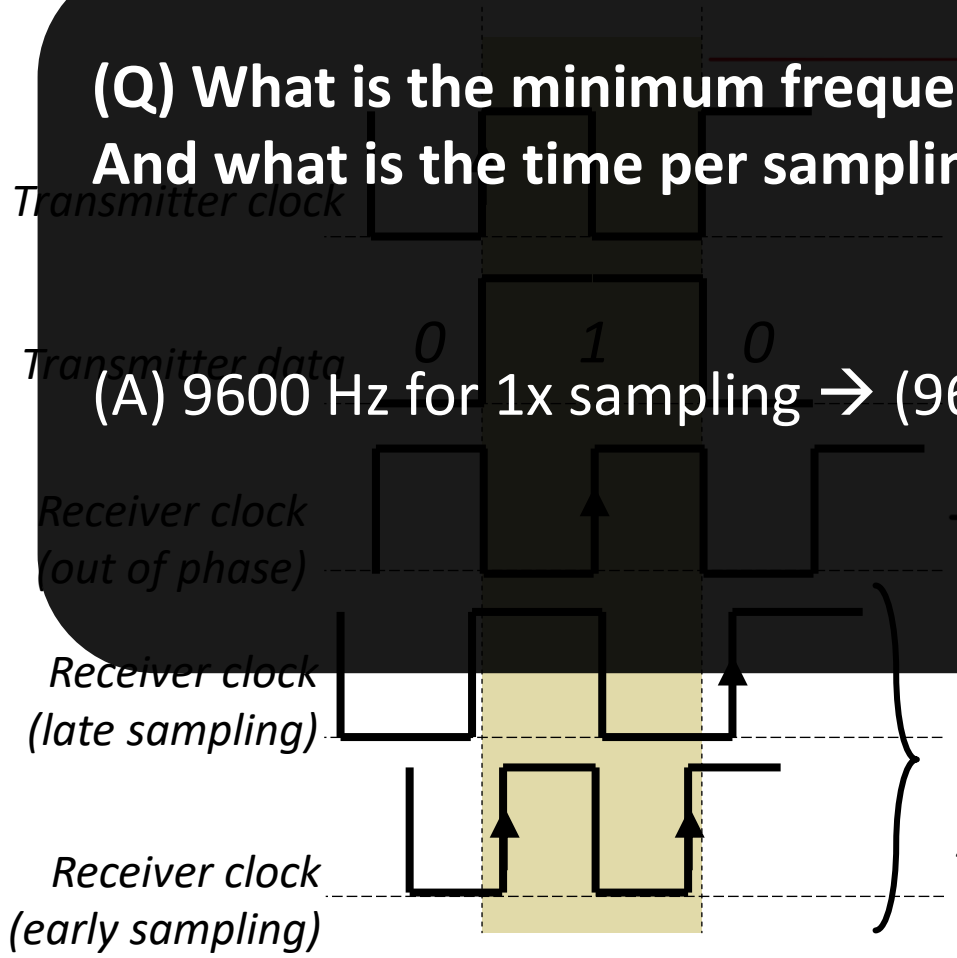
Median

*The **middle samples** (around the 8th or 9th sample in 16x oversampling)*

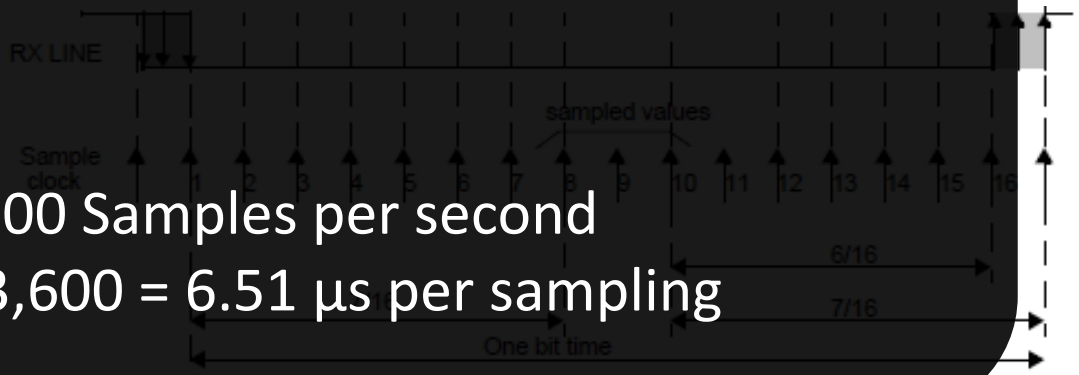
Oversampling in UART

- **UART is an ASYNCHRONOUS protocol. No shared clock.**

(Q) What is the minimum frequency for a system at 9600 baud with 16x Oversampling? And what is the time per sampling?



(A) 9600 Hz for 1x sampling → (9600).16 = 153.6 KHz
→ 153,600 Samples per second
→ 1/153,600 = 6.51 µs per sampling



At high frequencies, small deviations in the sampling signal frequency could corrupt the data.

Majority

*a **majority voting** technique to determine the bit's value*

Median

*The **middle samples** (around the 8th or 9th sample in 16x oversampling)*

UART in MSP430 – eUSCI Module

- Enhanced Universal Serial Communication Interface (eUSCI)
  - Supports multiple serial communication protocols
    - e.g., UART
    - e.g., serial peripheral interface (SPI)
    - e.g., inter-integrated Circuit (I2C)

- In MSP430FR6989
  - There are two implementations of eUSCI_A
    - eUSCI_A0
    - eUSCI_A1

    - Generally referred to as eUSCI_Ax

eUSCI Module - UCAxCTLW0 register

- Configuring the eUSCI_A in MSP430 for UART communication

UCAxCTLW0 register

UCPEN		UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC	UCSSELx							UCSWRST
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- UCWRST
  - Resetting the eUSCI_A
    - 0: eUSCI is operational.
    - 1: eUSCI is in reset.

Configuring and reconfiguring the eUSCI_A module should be done when UCSWRST is set (1) to avoid unpredictable behavior.

- UCSSEL
  - Determining the clock source for eUSCI_A

BRCLK

01: from ACLK

10: from SMCLK

eUSCI Module - UCAxCTLW0 register

- Configuring the eUSCI_A in MSP430 for UART communication

UCAxCTLW0 register

UCPEN		UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC	UCSSELx							UCSWRST
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- UCSYNC
  - Selecting between asynchronous (UART) and synchronous (SPI) operation
    - 0: asynchronous mode (UART).
    - 1: synchronous mode (SPI).

- UCMODEx
  - Determining the operation mode for eUSCI_A

00: UART Mode
01: Idle line (multiprocessor mode)
10: Address bit (multiprocessor mode)

eUSCI Module - UCAxCTLW0 register

- Configuring the eUSCI_A in MSP430 for UART communication

UCAxCTLW0 register

UCPEN		UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC	UCSSELx							UCSWRST
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- UCSPB
  - Determining the number of stop bits
    - 0: one stop bit.
    - 1: two stop bit.

- UC7BIT
  - Determining the character length (data)

0: 8-bit
1: 7-bit

eUSCI Module - UCAxCTLW0 register

- Configuring the eUSCI_A in MSP430 for UART communication

UCAxCTLW0 register

UCPEN		UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC	UCSSELx							UCSWRST
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- UCMSB
  - Determining the endianness (MSB or LSB first)
    - 0: LSB first.
    - 1: MSB first.

- UCPEN
  - Determining the parity being enable

eUSCI Module - Baud Rate Configuration

- **Determining baud rate for UART communication (without oversampling)**

UCAxMCTLW register

UCBRSx								UCBRFx							UCOS16
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

UCAxBRW register

UCBRx															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Step 1. *We need to determine the ratio between clock source and the desired baud rate*

from ACLK or SMCLK

$$N = f_{BRCLK} / (\text{baud rate})$$

Step 2. *When no oversampling is used then*

UCOS16 = 0 & UCBRFx = 0

0000 0000 0110 1000

Step 3. *Set the UCBRx (UCBRx = INT(N))*

e.g., f_{BRCLK} *= 1 MHz, baud rate = 9600*

UCBRx = 1M/9600 = 104.16

eUSCI Module - Baud Rate Configuration

- **Determining baud rate for UART communication (without oversampling)**

UCAxMCTLW register

UCBRSx								UCBRFx							UCOS16
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

0001 0001

UCAxBRW register

UCBRx															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fractional Portion of N	UCBRSx[1]		Fractional Portion of N	UCBRSx[1]
0.0000	0x00		0.5002	0xAA
0.0529	0x01		0.5715	0x6B
0.0715	0x02		0.6003	0xAD
0.0835	0x04		0.6254	0xB5
0.1001	0x08		0.6432	0xB6
0.1252	0x10		0.6667	0xD6
0.1430	0x20		0.7001	0xB7
0.1670	0x11		0.7147	0xBB
0.2147	0x21		0.7503	0xDD
0.2224	0x22		0.7861	0xED
0.2503	0x44		0.8004	0xEE
0.3000	0x25		0.8333	0xBF
0.3335	0x49		0.8464	0xDF
0.3575	0x4A		0.8572	0xEF
0.3753	0x52		0.8751	0xF7
0.4003	0x92		0.9004	0xFB
0.4286	0x53		0.9170	0xFD
0.4378	0x55		0.9288	0xFE

0000 0000 0110 1000

UCBRx = 1M/9600 = 104.16

?

eUSCI Module - Baud Rate Configuration

- **Determining baud rate for UART communication (with oversampling)**

UCAxMCTLW register

UCBRSx								UCBRFx							UCOS16
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

8 (1000)

UCAxBRW register

UCBRx															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

UCBRFx = INT([(6.51) – INT(6.51)] × 16)

Step 1. *We need to determine the ratio between clock source and the desired baud rate*

from ACLK or SMCLK

$N = f_{BRCLK} / (\text{baud rate})$

Step 2. *When oversampling is used then* → *UCOS16 = 1*

0000 0000 0000 0110

Step 3. *Set the UCBRx (UCBRx = INT(N/16))*

UCBRFx = INT([(N/16) – INT(N/16)] × 16)

e.g., f_{BRCLK} *= 1 MHz*, *baud rate = 9600*

*UCBRx = 1M/(16*9600) = 6.51*

UART in MSP430 – eUSCI Module

- **Determining baud rate for UART communication (with oversampling)**

UCAxMCTLW register

UCBRSx								UCBRFx							UCOS16
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

0001 0001

UCAxBRW register

UCBRx															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fractional Portion of N	UCBRSx[1]		Fractional Portion of N	UCBRSx[1]
0.0000	0x00		0.5002	0xAA
0.0529	0x01		0.5715	0x6B
0.0715	0x02		0.6003	0xAD
0.0835	0x04		0.6254	0xB5
0.1001	0x08		0.6432	0xB6
0.1252	0x10		0.6667	0xD6
0.1430	0x20		0.7001	0xB7
0.1670	0x11		0.7147	0xBB
0.2147	0x21		0.7503	0xDD
0.2224	0x22		0.7861	0xED
0.2503	0x44		0.8004	0xEE
0.3000	0x25		0.8333	0xBF
0.3335	0x49		0.8464	0xDF
0.3575	0x4A		0.8572	0xEF
0.3753	0x52		0.8751	0xF7
0.4003	0x92		0.9004	0xFB
0.4286	0x53		0.9170	0xFD
0.4378	0x55		0.9288	0xFE

0000 0000 0000 0110

*UCBRx = 1M/16*9600 = 6.51*

x 16

104.16

No oversampling

eUSCI Module - Baud Rate Configuration

- Determining baud rate for UART communication (with oversampling)

Recommended settings

BRCLK	Baud Rate	UCOS16	UCBRx	UCBRFx	UCBRSx[2]	TX Error[2] (%)		RX Error[2] (%)	
						neg	pos	neg	pos
32768	1200	1	1	11	0x25	-2.29	2.25	-2.56	5.35
32768	2400	0	13	-	0xB6	-3.12	3.91	-5.52	8.84
32768	4800	0	6	-	0xEE	-7.62	8.98	-21	10.25
32768	9600	0	3	-	0x92	-17.19	16.02	-23.24	37.3
1000000	9600	1	6	8	0x20	-0.48	0.64	-1.04	1.04
1000000	19200	1	3	4	0x2	-0.8	0.96	-1.84	1.84
1000000	38400	1	1	10	0x0	0	1.76	0	3.44
1000000	57600	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
1000000	115200	0	8	-	0xD6	-7.36	5.6	-17.04	6.96
1048576	9600	1	6	13	0x22	-0.46	0.42	-0.48	1.23
1048576	19200	1	3	6	0xAD	-0.88	0.83	-2.36	1.18
1048576	38400	1	1	11	0x25	-2.29	2.25	-2.56	5.35
1048576	57600	0	18	-	0x11	-2	3.37	-5.31	5.55
1048576	115200	0	9	-	0x08	-5.37	4.49	-5.93	14.92
4000000	9600	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
4000000	19200	1	13	0	0x84	-0.32	0.32	-0.64	0.48
4000000	38400	1	6	8	0x20	-0.48	0.64	-1.04	1.04
4000000	57600	1	4	5	0x55	-0.8	0.64	-1.12	1.76
4000000	115200	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
4000000	230400	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
4194304	9600	1	27	4	0xFB	0.11	0.1	0.33	0

$N = f_{BRCLK} / (baud\ rate)$

$UCBRx = INT(N/16)$

$UCBRx = INT(1048576/(16*9600))$
$= INT(6.82) = 6$

$UCBRFx = INT([(N/16) – INT(N/16)] \times 16)$
$= INT(0.82 \times 16) = 13$

$UCOS16 = 1$

$UCBRSx = map(N – INT(N))$
$= map(109.226 - 109)$
$= map(0.226) = 0x22$

eUSCI Module - Baud Rate Configuration

- Determining baud rate for UART communication (with oversampling)

UCAxCTLW0 register

UCPEN		UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC	UCSSELx							UCSWRST
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

UCAxMCTLW register

UCBRSx								UCBRFx							UCOS16
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

UCAxBRW register

UCBRx															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Parameter	Typical configuration
Baud Rate	9600 baud
Data size	8 bits
Bit ordering	LSB first
Parity bit	None
Stop bits	1 bit
Flow control	None

eUSCI Module - Reset Mode for Configuraiton

- Configuring the eUSCI_A in MSP430 for UART communication

UCAxCTLW0 register

UCPEN		UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC	UCSSELx							UCSWRST
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- UCWRST
  - Resetting the eUSCI_A
    - 0: eUSCI is operational.
    - 1: eUSCI is in reset.

Configuring and reconfiguring the eUSCI_A module should be done when UCSWRST is set (1) to avoid unpredictable behavior.

Enabling RESET

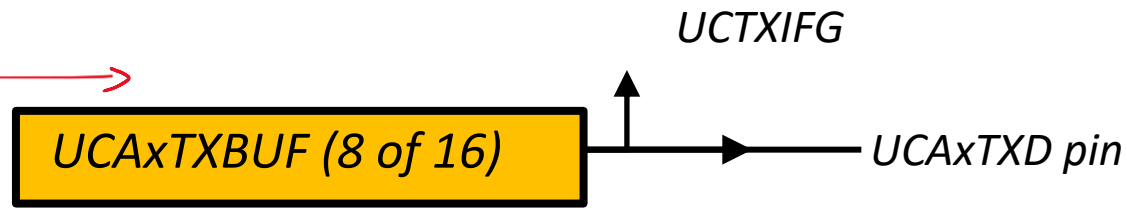
Disabling RESET

Some bits in UCAxCTLW0 register can be modified only when the eUSCI is in reset state.

```
void Initialize_UART(void){
// ….
// Main configuration register
UCA1CTLW0 = UCSWRST; // Engage reset; change all the fields to zero
// Most fields in this register, when set to zero, correspond to the
// popular configuration
UCA1CTLW0 |= UCSSEL_2; // Set clock to SMCLK
// Configure the clock dividers and modulators (and enable oversampling)
UCA1BRW = 6; // divider
// Modulators: UCBRF = 8 = 1000 --> UCBRF3 (bit #3)
// UCBRS = 0x20 = 0010 0000 = UCBRS5 (bit #5)
UCA1MCTLW = UCBRF3 | UCBRS5 | UCOS16;
// Exit the reset state
UCA1CTLW0 &= ~UCSWRST;
}
```

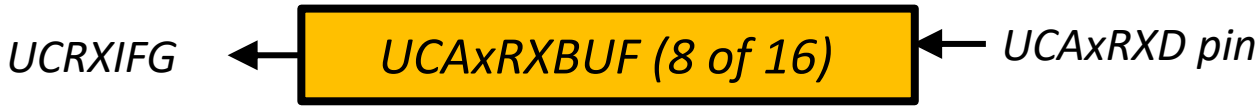
eUSCI Module - UART Transmission/Reception

- 16-bit Register for data transmission/reception
  - Only 8 bits (lower byte) are used
    - the data size is only 8 bits.
- UCTXIFG is a transmit interrupt flag in the UCAxIFG register.
  - 1: transmission buffer is empty
  - 0: transmission in progress




```
void uart_write_char(unsigned char ch){ // transmitting a byte of data (a character)
        while((UCA1IFG & UCTXIFG) == 0){}  // Wait here. Tx in progress
        // Exit while loop. Transmission is complete. Buffer is now empty.
        UCA1TXBUF = ch;
}
```

- UCRXIFG is a receive interrupt flag in the UCAxIFG register
  - 1: data received
  - 0: no data received



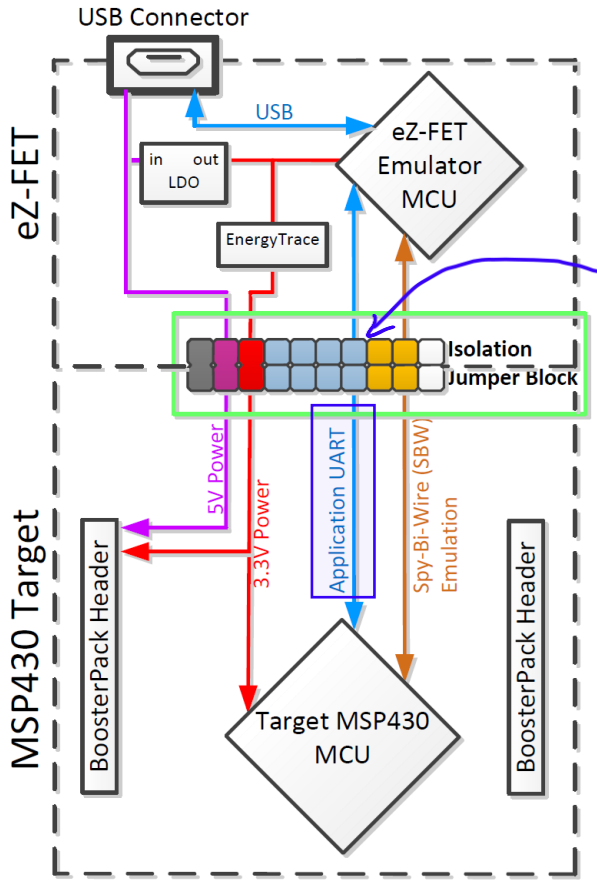

```
unsigned char uart_read_char(void){ // receiving a byte of data (a character)
        unsigned char ch;
        if((UCA1IFG & UCRXIFG) == 0) { // no data (character) received
                return 0;}
        // Otherwise, copy the received byte (character) and return it
        ch = UCA1RXBUF;
        return ch;
}
```

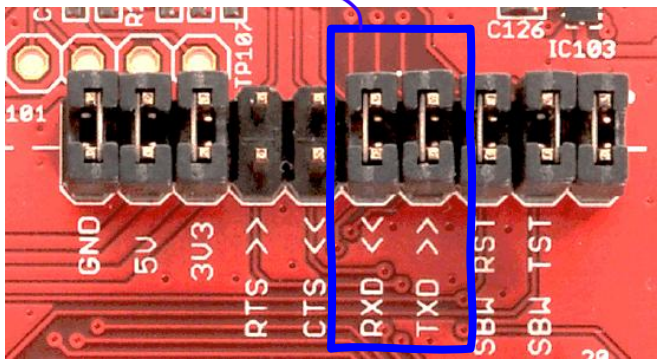
eUSCI Module - UART Connection

- Backchannel UART (Application UART)
  - Connecting UART to USB port to avoid requiring a separate external adapter (UART-to-USB)
  - Connected to the onboard FET (Flash Emulation Tool)

    debugging, logging, and simple data exchange between the microcontroller and a host PC

  - Connection is done by the jumper pins
    - can be removed to disconnect the UART signals between the MSP430 and the FET



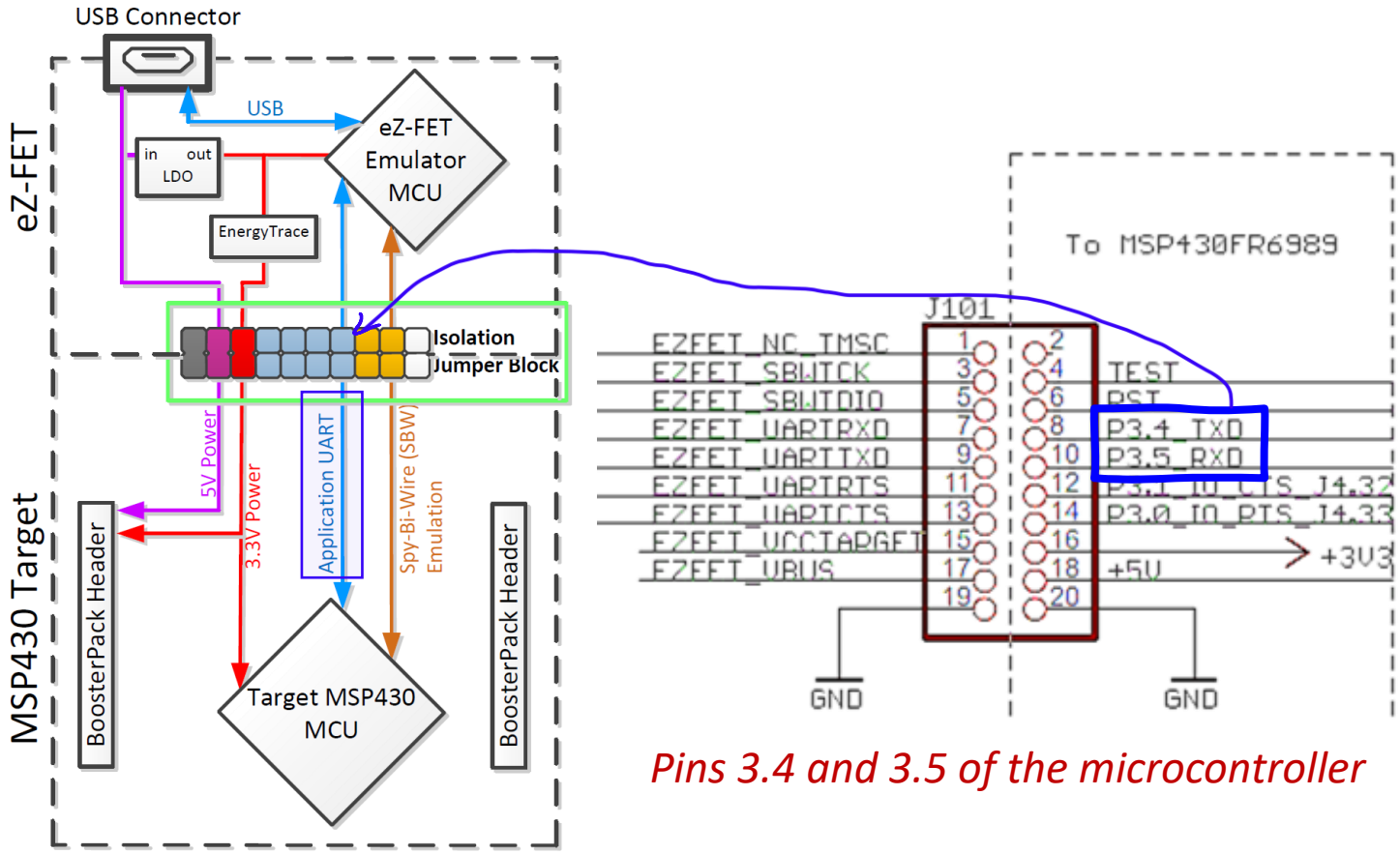


eUSCI Module - UART Connection

- Backchannel UART (Application UART)
  - Connecting UART to USB port to avoid requiring a separate external adapter (UART-to-USB)
  - Connected to the onboard FET (Flash Emulation Tool)

    debugging, logging, and simple data exchange between the microcontroller and a host PC

  - Connection is done by the jumper pins
    - can be removed to disconnect the UART signals between the MSP430 and the FET



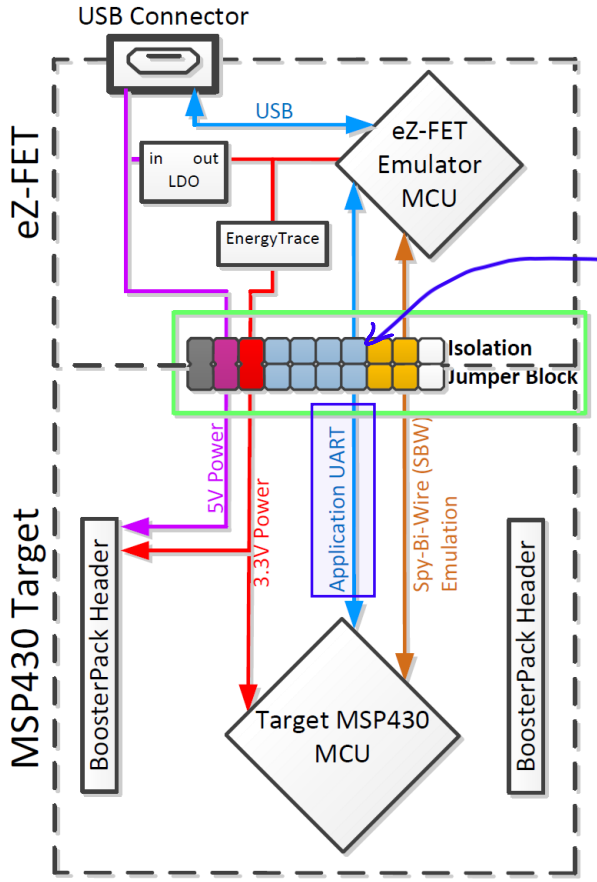
Pins 3.4 and 3.5 of the microcontroller

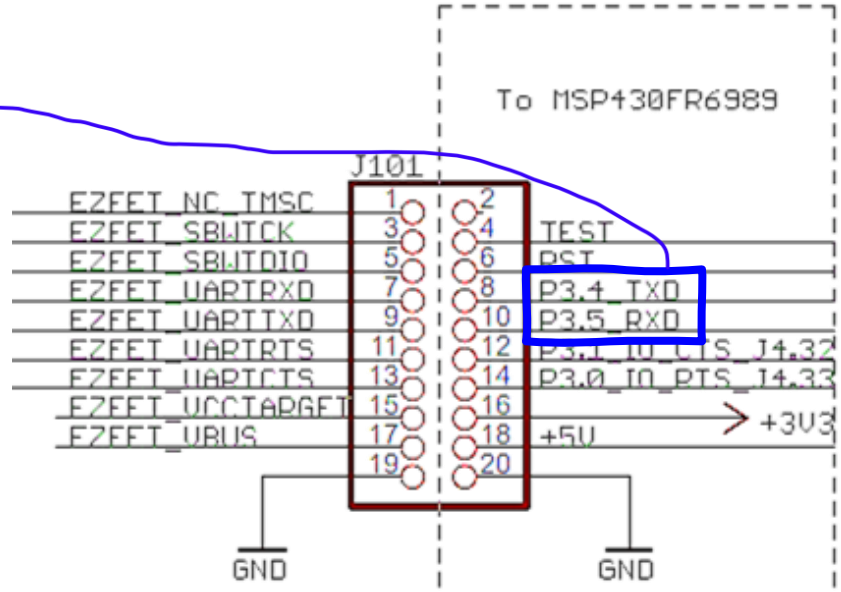
eUSCI Module - UART Connection

- Backchannel UART (Application UART)
  - Connecting UART to USB port to avoid requiring a separate external adapter (UART-to-USB)

Pin number	x	Function	P3SEL1.x	P3SEL0.x
		P3.4	0	0
40	4	UCA1TXD	0	1
		...	...	...
		P3.5	0	0
41	5	UCA1RXD	0	1
		...	...	...

Enable the UART functionality on the relevant pins using the P3SEL registers.





Pins 3.4 and 3.5 of the microcontroller

UART in MSP430 - eUSCI A0 and A1

- In MSP430FR6989
  - There are two implementations of eUSCI_A
    - eUSCI_A0
    - eUSCI_A1
  - Generally referred to as eUSCI_Ax

USCI function	Pin number	GPIO function
UCA0TXD	100	P4.2
	51	P2.0
UCA0RXD	1	P4.3
	50	P2.1
UCA1TXD	87	P5.4
	40	P3.4
UCA1RXD	88	P5.5
	41	P3.5

There is a second set of pins (P4.2 and P4.3), that can be used

There is a second set of pins (P5.4 and P5.5), that can be used

*transmit data pin for the **USCI A0***

*receive data pin for the **USCI A0***

*transmit data pin for the **USCI A1***

*receive data pin for the **USCI A1***

```
// Divert pins to UART functionality
P3SEL1 &= ~(BIT4|BIT5);
P3SEL0 |= (BIT4|BIT5);
```

UART Programming in MSP430

- **Configuration of UART**

UCAxCTLW0 register

UCPEN		UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC	UCSSELx						UCSWRST
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

UCAxMCTLW register

UCBRSx								UCBRFx							UCOS16
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

UCAxBRW register

UCBRx															
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

```
// Configure UART to the popular configuration
// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit, no flow control
// Initial clock: SMCLK @ 1.048 MHz with oversampling
void Initialize_UART(void){
        // Divert pins to UART functionality
        P3SEL1 &= ~(BIT4|BIT5);
        P3SEL0 |= (BIT4|BIT5);
        UCA1CTLW0 = UCSWRST;
        // Use SMCLK clock; leave other settings default
        UCA1CTLW0 |= UCSSEL_2;
        // Configure the clock dividers and modulators
        // UCBR=6, UCBRF=13, UCBRS=0x22, UCOS16=1 (oversampling)
        UCA1BRW = 6;
        UCA1MCTLW = UCBRS5|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;
        // Exit the reset state (so transmission/reception can begin)
        UCA1CTLW0 &= ~UCSWRST;
}
```

BRCLK	Baud Rate	UCOS16	UCBRx	UCBRFx	UCBRSx[2]	TX Error[2] (%)		RX Error[2] (%)	
						neg	pos	neg	pos
1048576	9600	1	6	13	0x22	-0.46	0.42	-0.48	1.23

Interrupt for UART in MSP430

- Interrupt can be used to handle TX and RX separately

UCAxIE register

														UCTXIE	UCRXIE
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Transmit interrupt enable *(when '1', interrupt will be triggered when the transmit buffer is ready to accept new data)*
Receive interrupt enable *(when '1', interrupt will be triggered when new data is received and ready to be read from the receive buffer.)*

USCI_A1_VECTOR *is the interrupt vector associated with the USCI module*

- The flag corresponding to UCTXIE, UCTXFG will be automatically cleared
  - when a data is written to the transmit buffer.
- Similarly, the flag corresponding to UCRXIE, UCRXFG will be automatically cleared
  - when the data is read from the receive buffer.

Example 1

- What is this code for?

```
void main() {
        WDTCTL = WDTPW | WDTHOLD;
        PM5CTL0 &= ~LOCKLPM5;

        unsigned char temp;

    Initialize_UART();
    while(1){
        // Read input
        temp = uart_read_char();
        // Display if it is not 0
        if(temp != 0) {
                uart_write_char(temp);
        }
    }
}
```

Example 1

- **What is this code for?**

- *Reads a single character from the UART receive buffer.*
- *checks if the received data (temp) is not zero (validation).*
- *sends the character back via UART, possibly for **echoing** or further processing.*

```
void main() {
        WDTCTL = WDTPW | WDTHOLD;
        PM5CTL0 &= ~LOCKLPM5;

        unsigned char temp;

    Initialize_UART();
    while(1){
        // Read input
        temp = uart_read_char();
        // Display if it is not 0
        if(temp != 0) {
                uart_write_char(temp);
        }
    }
}
```

Example 2

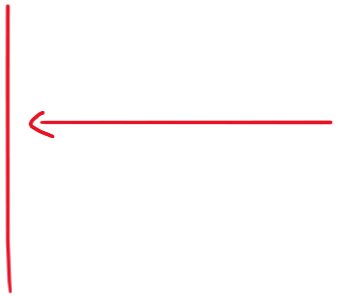
- **What is this code for?**

```
void main() {
   WDTCTL = WDTPW | WDTHOLD;
   PM5CTL0 &= ~LOCKLPM5;

   unsigned char temp = 0;
   volatile unsigned int i;

   Initialize_UART();
   while(1){
      uart_write_uint8(temp++);  //Auto-increment operator to rollover
                  // Move to next line
      uart_newline();
                  // Add a short delay
      for(i=20000; i>0; i--){}
   }
}

void uart_write_uint8(unsigned char n){
   uart_write_char(48 + n/100);
   n = n%100;
   uart_write_char(48 + n/10);
   n = n%10;
   uart_write_char(48 + n);
}

void uart_newline(){
   uart_write_char(10);                                    // Line Feed
   uart_write_char(13);                                    // Carriage return
}
```

Example 2

- **What is this code for?**

- *Sends the current value of temp (starting from 0) over UART.*
- *After sending, temp is incremented (up to 255 and then roll over from 255 back to 0).*
- *Per each sending, a newline() is called. Likely it is "\n" or "\r".*
- *Add manual loop delay per each transmission.*



```
void main() {
   WDTCTL = WDTPW | WDTHOLD;
   PM5CTL0 &= ~LOCKLPM5;

   unsigned char temp = 0;
   volatile unsigned int i;

   Initialize_UART();
   while(1){
      uart_write_uint8(temp++);  //Auto-increment operator to rollover
                  // Move to next line
      uart_newline();
                  // Add a short delay
      for(i=20000; i>0; i--){}
   }
}

void uart_write_uint8(unsigned char n){
   uart_write_char(48 + n/100);
   n = n%100;
   uart_write_char(48 + n/10);
   n = n%10;
   uart_write_char(48 + n);
}

void uart_newline(){
   uart_write_char(10);                    // Line Feed
   uart_write_char(13);                    // Carriage return
}
```

Example 2

- **What is this code for?**

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0 ^@ NUL NULL	1 ^A SOH START OF HEADING	2 ^B STX START OF TEXT	3 ^C ETX END OF TEXT	4 ^D EOT END OF TRANSM.	5 ^E ENQ ENQUIRY	6 ^F ACK ACKNOWL-EDGE	7 ^G BEL BELL	8 ^H BS BACKSP.	9 ^I HT CHARACT. TAB'TION	10 ^J LF LINE FEED	11 ^K VT LINE TAB'TION	12 ^L FF FORM FEED	13 ^M CR CARRIAGE RETURN	14 ^N SO SHIFT OUT	15 ^O SI SHIFT IN
1	16 ^P DLE DATALINK ESCAPE	17 ^Q DC1 DEVICE CONTROL1	18 ^R DC2 DEVICE CONTROL2	19 ^S DC3 DEVICE CONTROL3	20 ^T DC4 DEVICE CONTROL4	21 ^U NAK NEG.ACK-NOWLEDGE	22 ^V SYN SYNCHR. IDLE	23 ^W ETB END OF TRANS.	24 ^X CAN CANCEL	25 ^Y EM END OF MEDIUM	26 ^Z SUB SUBS-TITUTE	27 ^[ESC ESCAPE	28 ^\ FS INFO. SEP. 4	29 ^] GS INFO. SEP. 3	30 ^^ RS INFO. SEP. 2	31 ^_ US INFO. SEP. 1
2	32 SPACE	33 excl ! EXCLAM. MARK	34 quot " QUOT. MARK	35 num # NUMBER SIGN	36 dollar $ DOLLAR SIGN	37 percnt % PERCENT SIGN	38 amp & AMPER-SAND	39 apos ' APOS-TROPHE	40 lpar (LEFT PAREN.	41 rpar) RIGHT PAREN.	42 ast * ASTERISK	43 plus + PLUS SIGN	44 comma , COMMA	45 - HYPHEN-MINUS	46 period . FULL STOP	47 sol / SOLIDUS
3	48 0 DIGIT ZERO	49 1 DIGIT ONE	50 2 DIGIT TWO	51 3 DIGIT THREE	52 4 DIGIT FOUR	53 5 DIGIT FIVE	54 6 DIGIT SIX	55 7 DIGIT SEVEN	56 8 DIGIT EIGHT	57 9 DIGIT NINE	58 colon : COLON	59 semi ; SEMI-COLON	60 lt < LS.-THAN SIGN	61 equals = EQUALS SIGN	62 gt > GR.-THAN SIGN	63 quest ? QUEST-ION MARK
4	64 commat @ COMM'IAL AT	65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
5	80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 lsqb [LEFT SQ. BRACKET	92 bsol \ REVERSE SOLIDUS	93 rsqb] RT. SQR. BRACKET	94 hat ^ CIRCUM'X ACCENT	95 lowbar _ LOW LINE
6	96 grave ` GRAVE ACCENT	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
7	112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 { L. CURLY BRACKET	124 | VERTICAL LINE	125 } R. CURLY BRACKET	126 ~ TILDE	127 ^? DEL DELETE

```
void main() {
   WDTCTL = WDTPW | WDTHOLD;
   PM5CTL0 &= ~LOCKLPM5;

   unsigned char temp = 0;
   volatile unsigned int i;

   Initialize_UART();
   while(1){
      uart_write_uint8(temp++);  //Auto-increment operator to rollover
                  // Move to next line
      uart_newline();
                  // Add a short delay
      for(i=20000; i>0; i--){}
   }
}

void uart_write_uint8(unsigned char n){
   uart_write_char(48 + n/100);
   n = n%100;
   uart_write_char(48 + n/10);
   n = n%10;
   uart_write_char(48 + n);
}

void uart_newline(){
   uart_write_char(10);                    // Line Feed
   uart_write_char(13);                    // Carriage return
}
```

Example 3

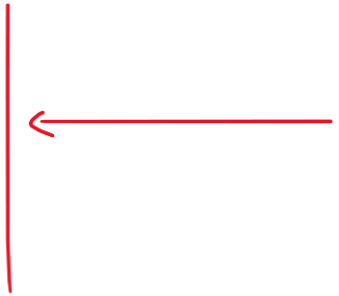
- What is this code for?

```
void main() {
        WDTCTL = WDTPW | WDTHOLD;
        PM5CTL0 &= ~LOCKLPM5;
        Initialize_UART();

        \\ Character array
        unsigned char data[] = "Hello World!";
        \\ Pointer to the array
        unsigned char *i = &data;

        \\ Print if the element is not NULL
        while(*i != '\0'){

                uart_write_char(*i);
                i++;
        }
        while(1);                              \\ Infinite loop
}
```

Example 3

- **What is this code for?**

- *Prints each character of the string "Hello World!" one by one over the UART.*
- *The pointer i is incremented to traverse through the characters of the string.*
- *Once the NULL character "\0", is encountered, the loop exits, and the program enters the infinite loop*



```
void main() {
        WDTCTL = WDTPW | WDTHOLD;
        PM5CTL0 &= ~LOCKLPM5;
        Initialize_UART();

        \\ Character array
        unsigned char data[] = "Hello World!";
        \\ Pointer to the array
        unsigned char *i = &data;

        \\ Print if the element is not NULL
        while(*i != '\0'){

                uart_write_char(*i);
                i++;
        }
        while(1);                          \\ Infinite loop
}
```

Thank You!

Questions?

Email: hadi.mardanikamali@ucf.edu
UCF HEC 435 (407) 823 – 0764
https://www.ece.ucf.edu/~kamali/

HAVEN Research Group
https://haven.ece.ucf.edu/

HAVEN ASSURANCE VERIFICATION

UCF UNIVERSITY OF CENTRAL FLORIDA