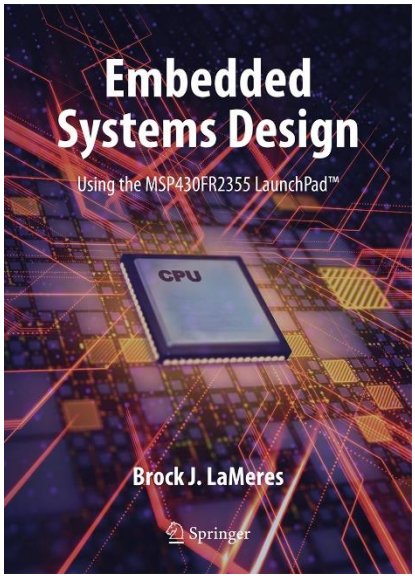
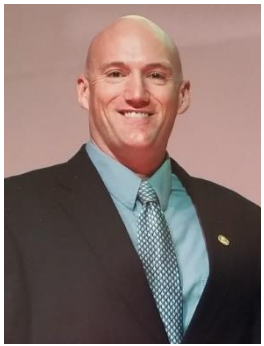
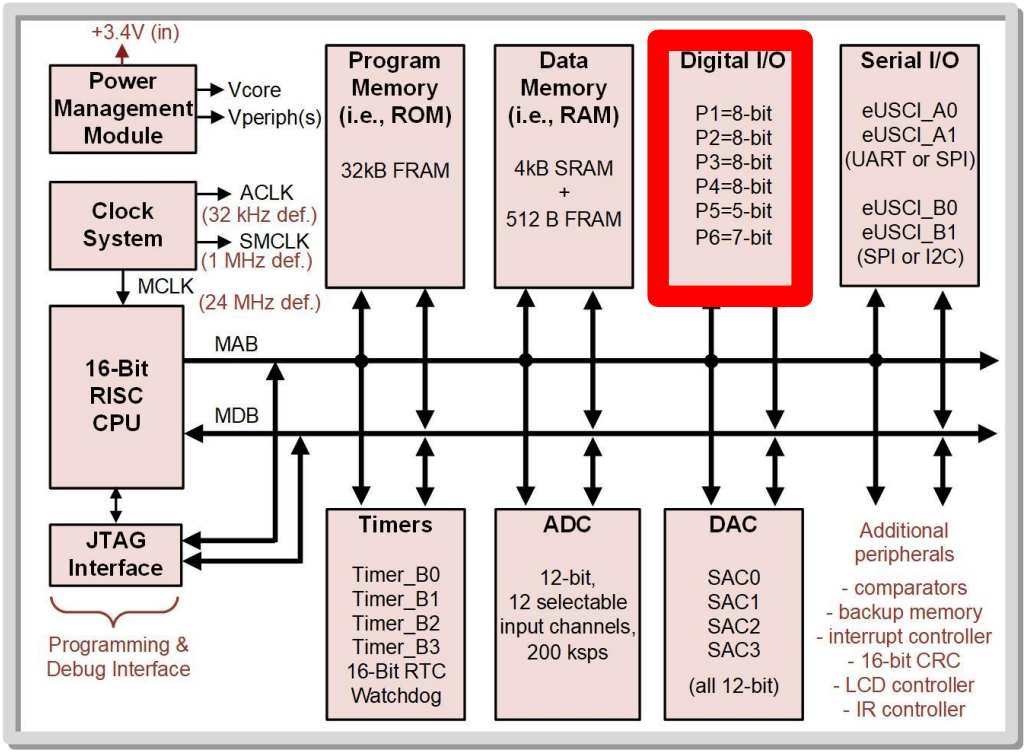
CHAPTER 14: SERIAL COMMUNICATIONS IN C

14.1 SERIAL COMMUNICATIONS OVERVIEW



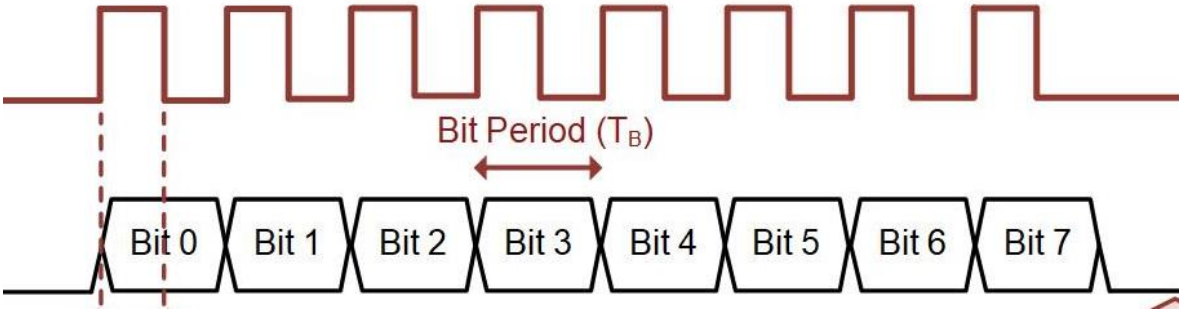


- **Parallel Communication** – How we send information as a full word using the MCU ports. a full word.



- **Serial Communication** – uses a single wire to send information between a transmitter (Tx) and a receiver (Rx) as a sequence of bits.

- Serial communication allows information to be transmitted using less pins compared to parallel communication, but at a slower overall information transmission rate due to only sending one bit at a time as opposed to a full word.



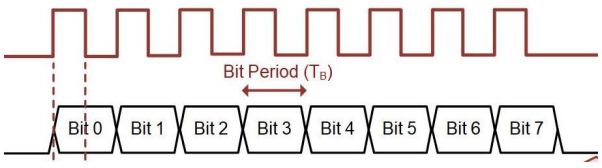
- **Serial Communications on MCU's**

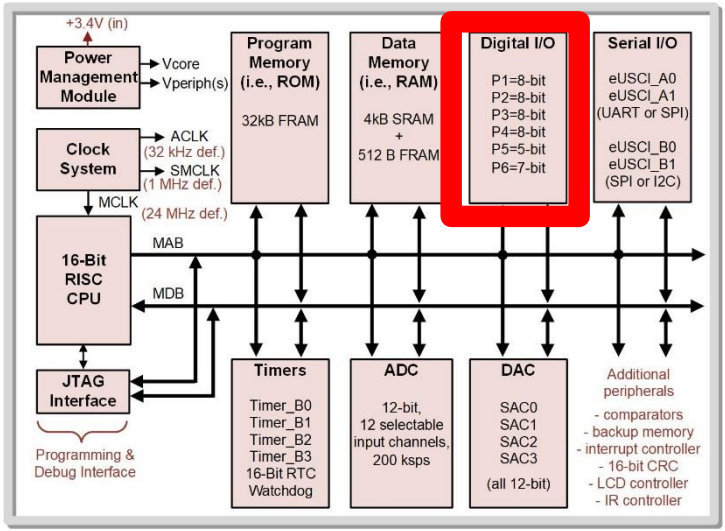
- **Serial Communications on MCU's**

  - Method 1 – "Bit Banging"

  - use BIC and BIS instructions to send the desired pattern out of a port pin.

  - Timer IRQs are used to set the data and clock periods.





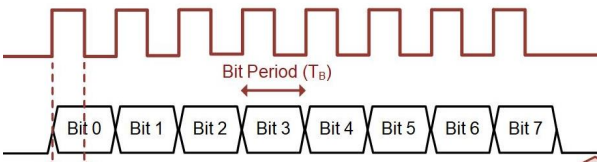
- **Serial Communications on MCU's**

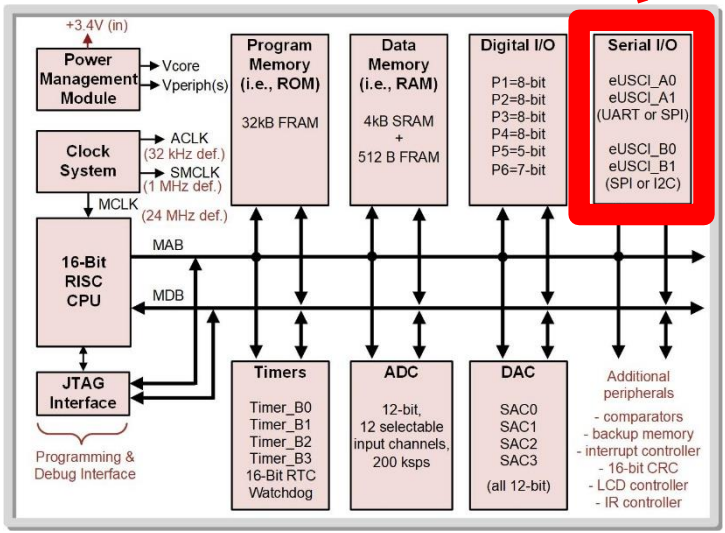
  - Method 1 – "Bit Banging"

  - Method 2 – "Built in Serial Peripherals"

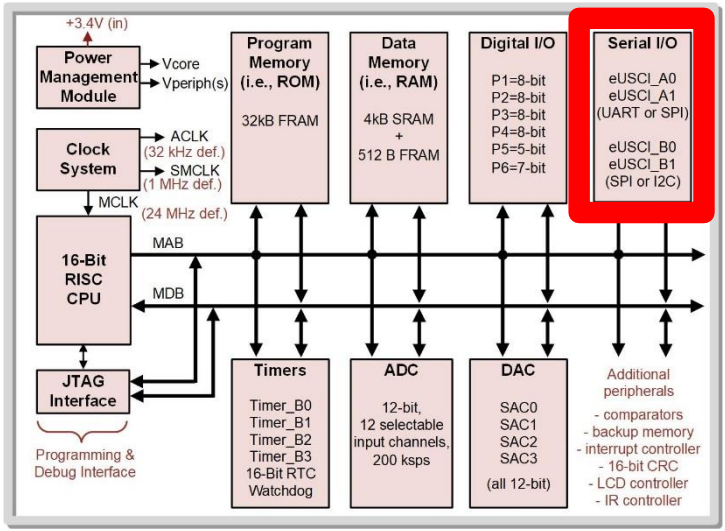
    - enhanced Universal Serial Communication Interfaces (eUSCI).

    - The MSP430FR2355 contains four eUSCI peripherals (eUSCI_A0, eUSCI_A1, eUSCI_B0, eUSCI_B1)





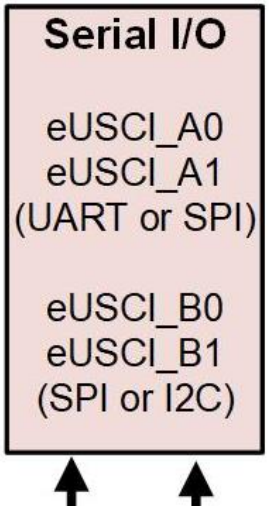
- **MSP430FR2355 serial communication standards:**
  - Universal Asynchronous Receiver/Transmitter (UART)
  - Serial Peripheral Interface (SPI)
  - Inter-integrated circuits (I2C) protocol

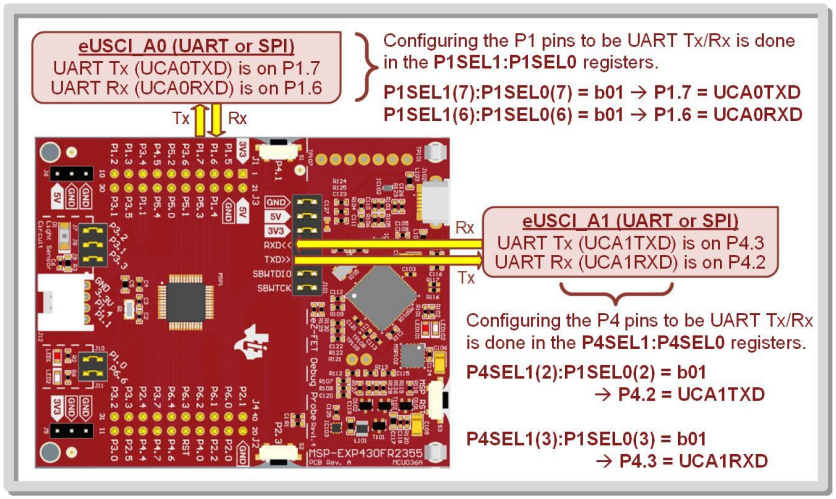


- **Serial Communications on MCU's**

  - Method 1 – "Bit Banging"

  - Method 2 – "Built in Serial Peripherals"





- **Parallel-to-Serial Concept (Tx)**

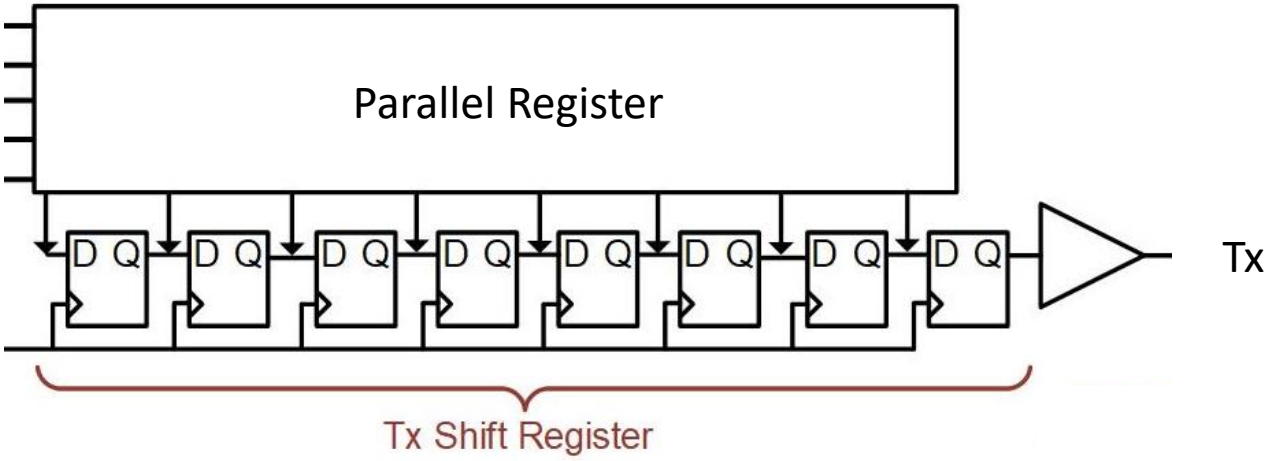


Image Courtesy of https://neodem.wp.horizon.ac.uk/

- **Serial-to-Parallel Concept (Tx)**

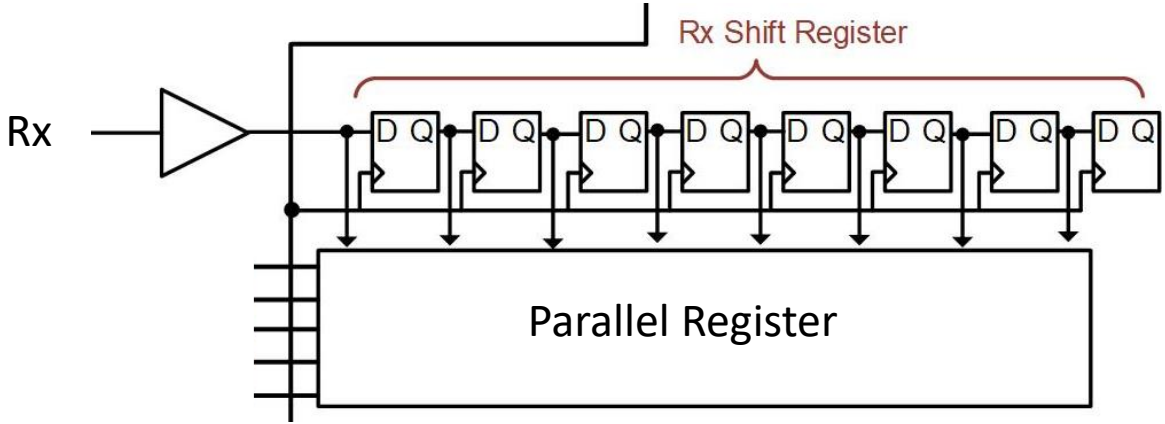


Image Courtesy of https://neodem.wp.horizon.ac.uk/

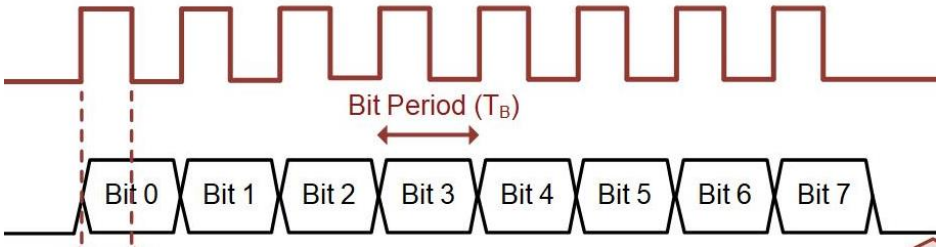
- **Serial Communications on MCU's**

  - Learning….

  Step 1 – you need to know the *standard*.

  Step 2 – decide whether to bit bang or use peripheral.

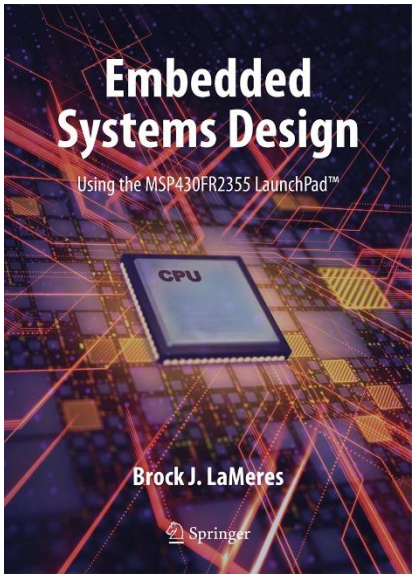
We will use the peripherals. After you see the serial busses in action, bit banging will make more sense.



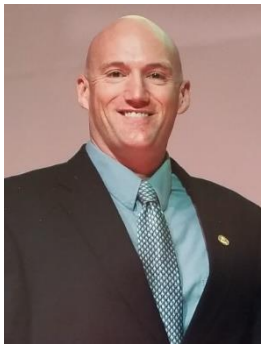
Embedded Systems Design

Chapter 14: Serial Communications in C

14.1 Serial Communications Overview

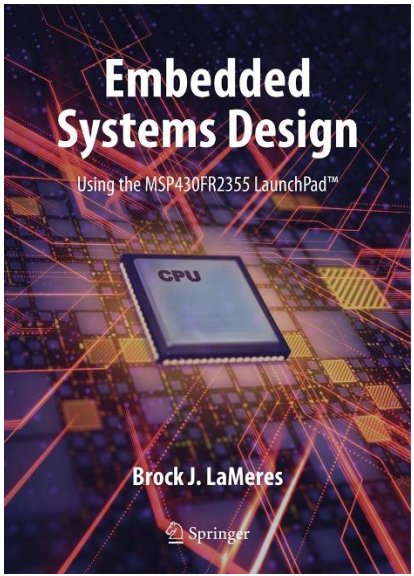


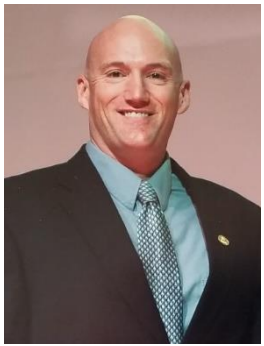
www.youtube.com/c/DigitalLogicProgramming_LaMeres



CHAPTER 14: SERIAL COMMUNICATIONS IN C

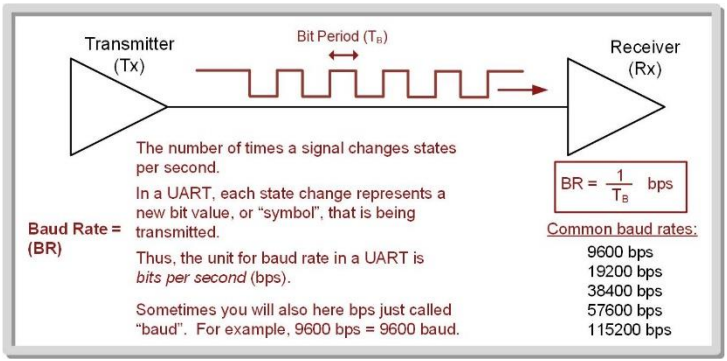
14.1.1 THE UART STANDARD





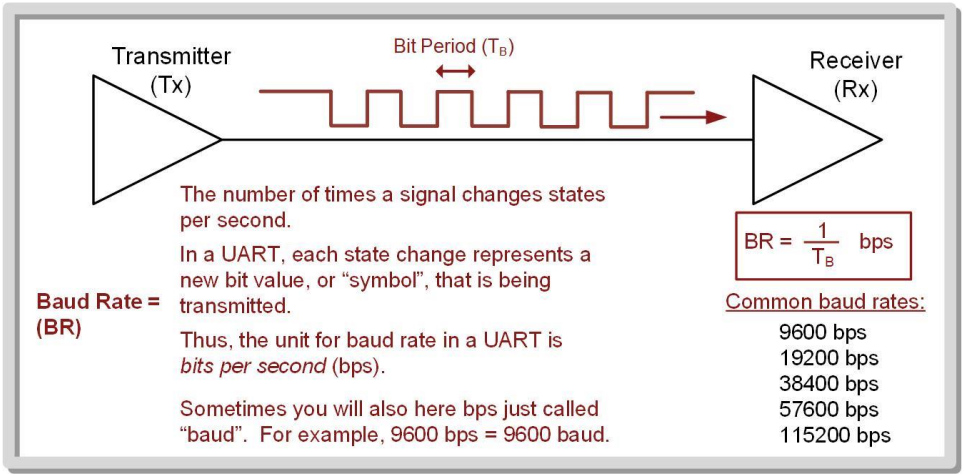
14.1.1 THE UART STANDARD

- **Universal asynchronous receiver/transmitter** – an approach to serial communication between two devices in which neither device shares a common clock.
- Each device only uses lines for data.
- Has a predefined protocol for how data is framed, how data packets start and stop, and how the receiver oversamples the incoming data to recover the information sent.



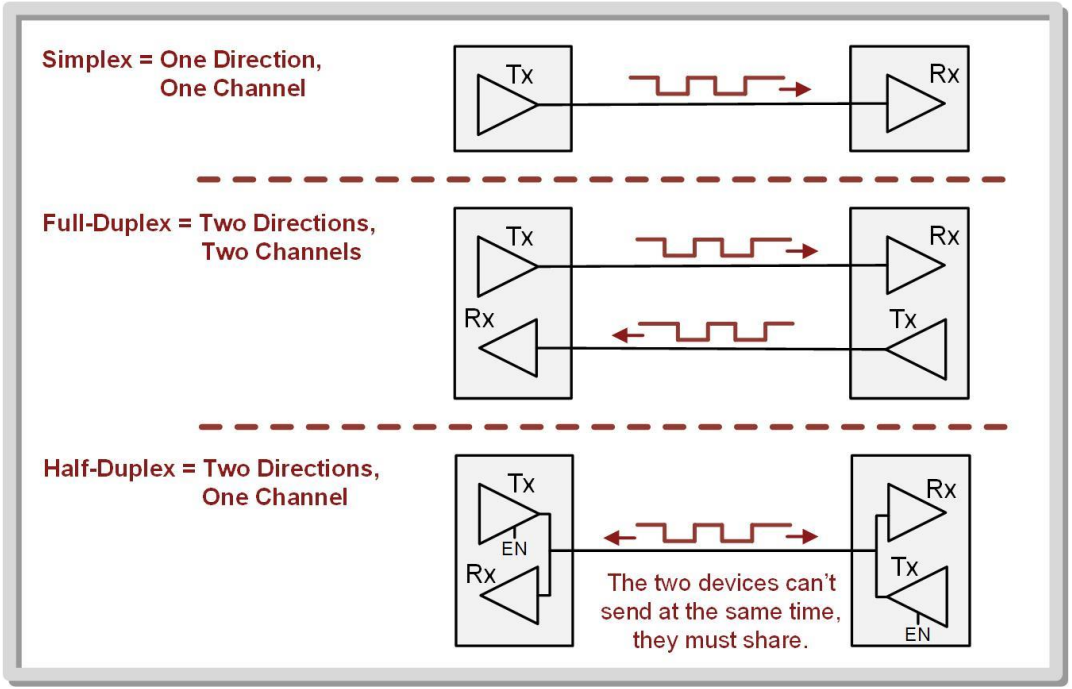
14.1.1 THE UART STANDARD

- **Baud rate (BR)** – the common data rate between Tx and Rx; the fastest rate at which the data line changes states.
- The baud rate and bit period (T_B) are related by $BR = 1/T_B$.
- Baud rate must be set manually in both the Tx and Rx prior to data transmission (typical range is 9600 to 115200 baud).



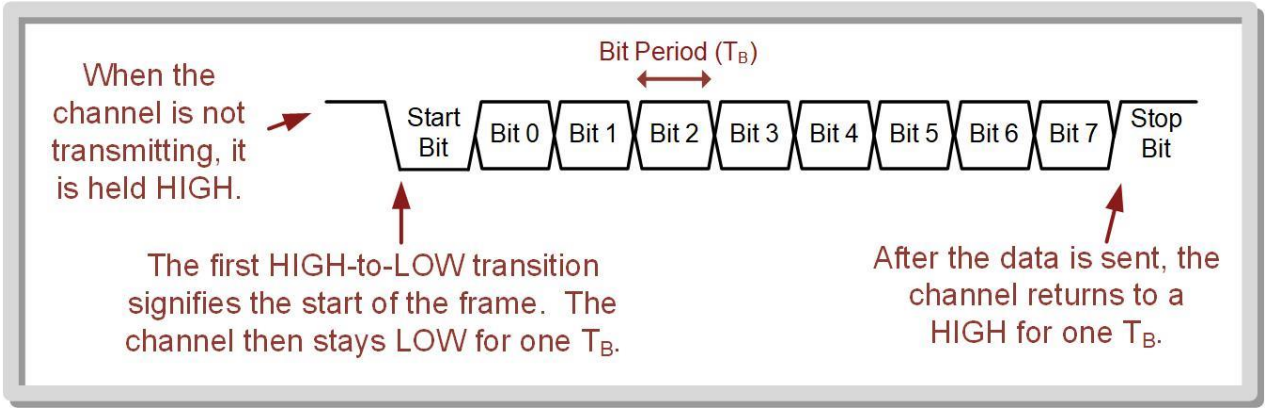
14.1.1 THE UART STANDARD

- **Link** – when two devices are connected together to transfer information.



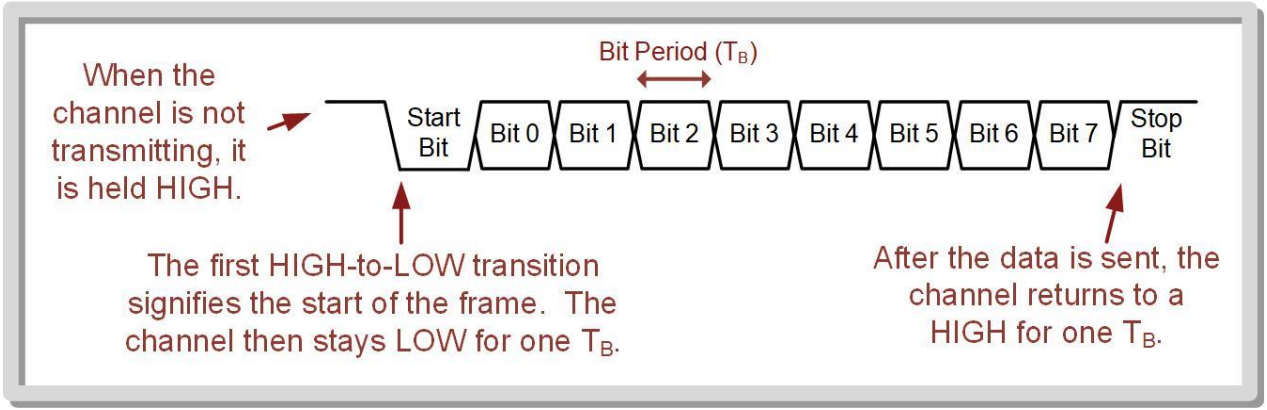
14.1.1 THE UART STANDARD

- **Framing** – the term used to describe how the bits are arranged in the UART serial bit sequence.



14.1.1 THE UART STANDARD

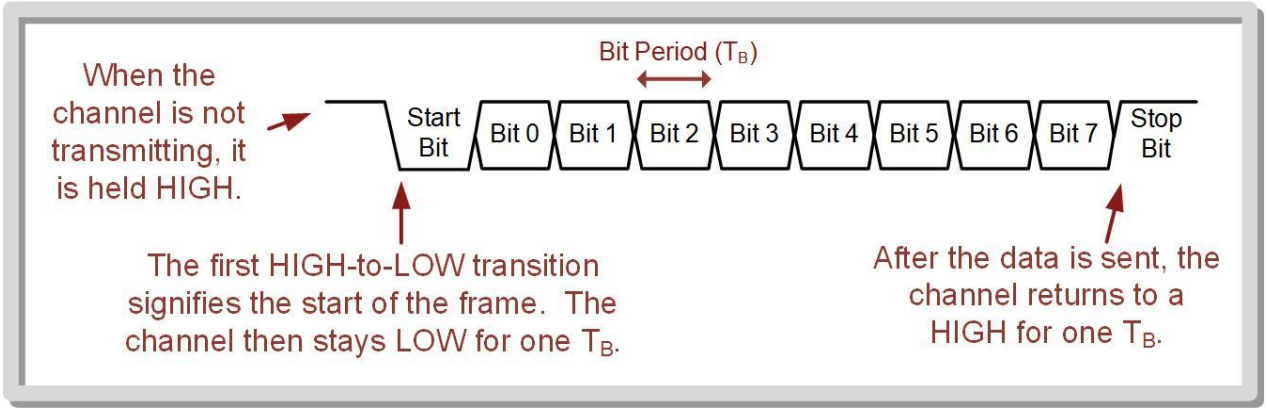
- **Framing** – the term used to describe how the bits are arranged in the UART serial bit sequence.



- Having the Tx drive a HIGH while not transmitting tells the Rx that it is still powered on.

14.1.1 THE UART STANDARD

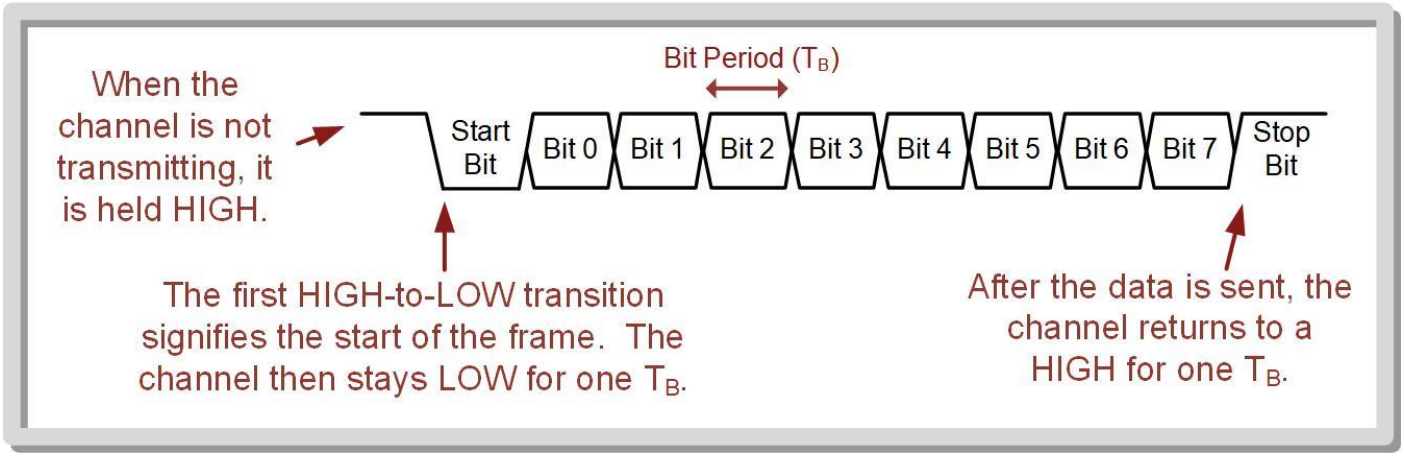
- **Framing** – the term used to describe how the bits are arranged in the UART serial bit sequence.



- **Start bit** – initial symbol
- To start a data transmission, the Tx drives the line LOW and holds it there for one T_B.

14.1.1 THE UART STANDARD

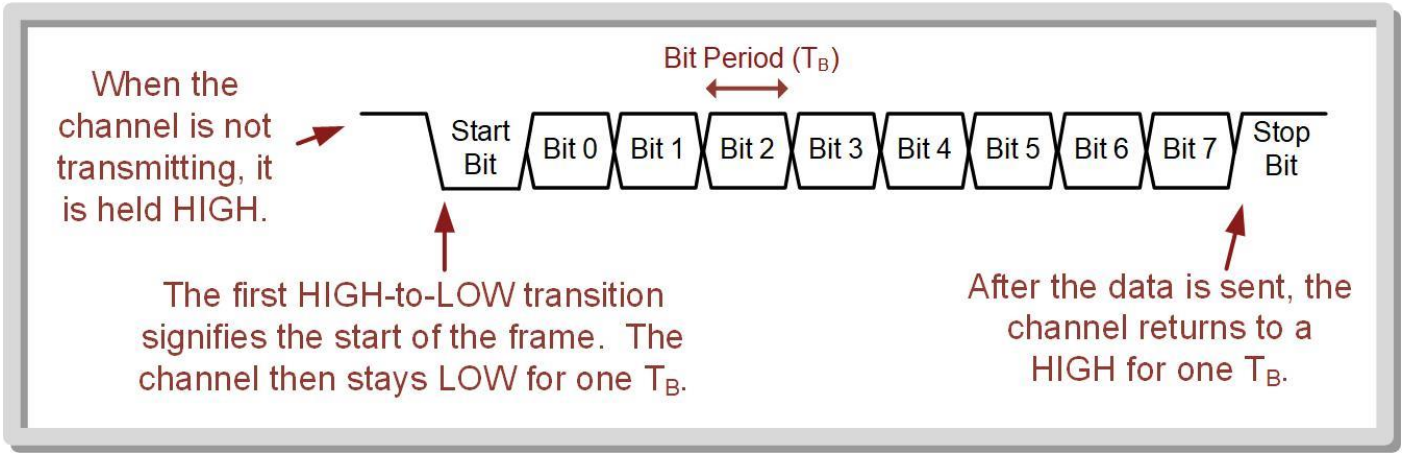
- The data word is transmitted bit-by-bit, typically starting with the LSB.



14.1.1 THE UART STANDARD

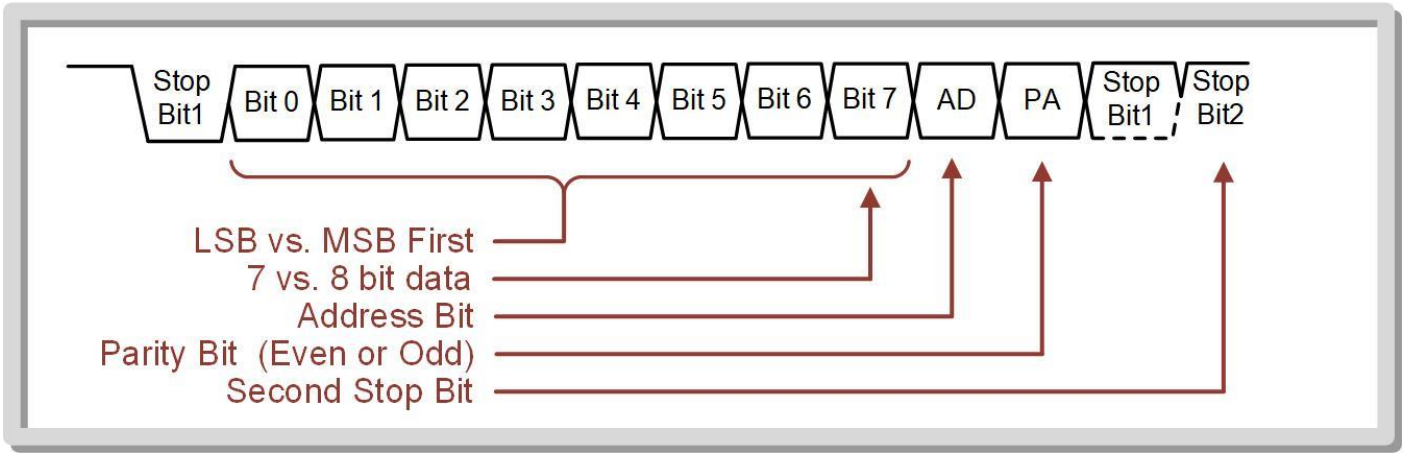
- **Stop bit** – last symbol

- After the bits have been transmitted, the Tx drives the line HIGH for one T_B to represent the end of the sequence.



14.1.1 THE UART STANDARD

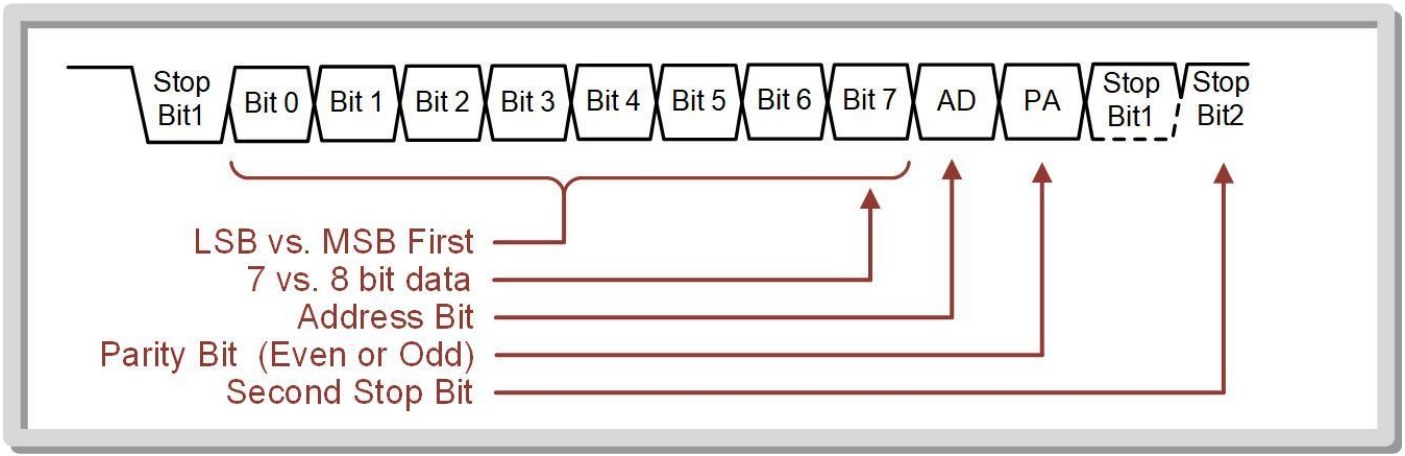
- The most common options that are configurable on the MSP430 are swapping the order in which the bits are sent, changing the data size between 7-bits and 8-bits, adding on **address bit (AD)**, adding a **parity bit (PA)**, and adding a second stop bit.



14.1.1 THE UART STANDARD

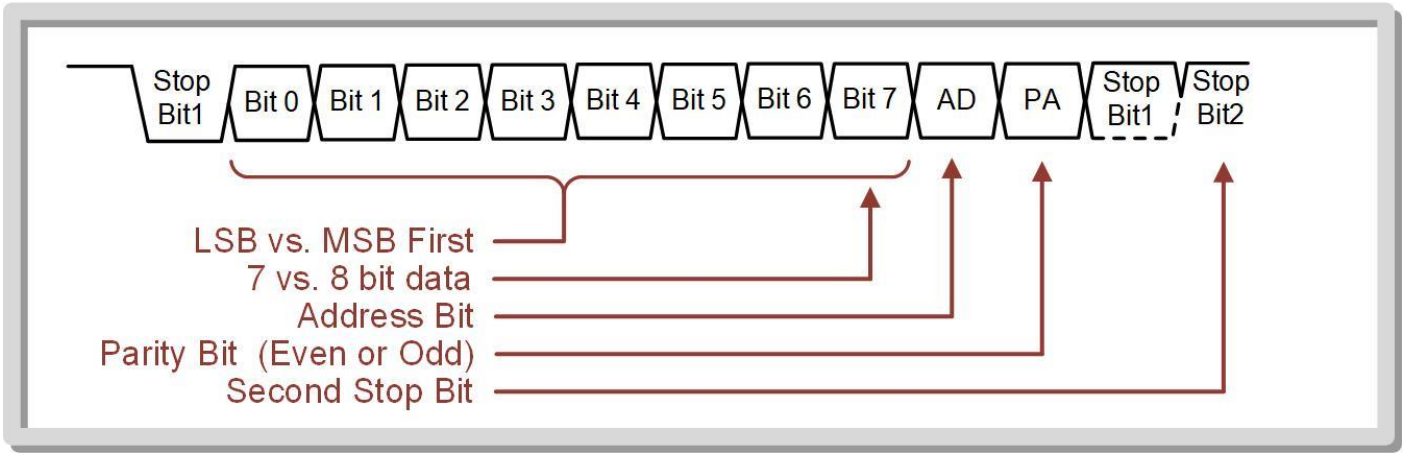
- When using a **transmit address**, two frames are sent, one for data and one for address.

- The AD bit signifies whether the next frame is the address or data.



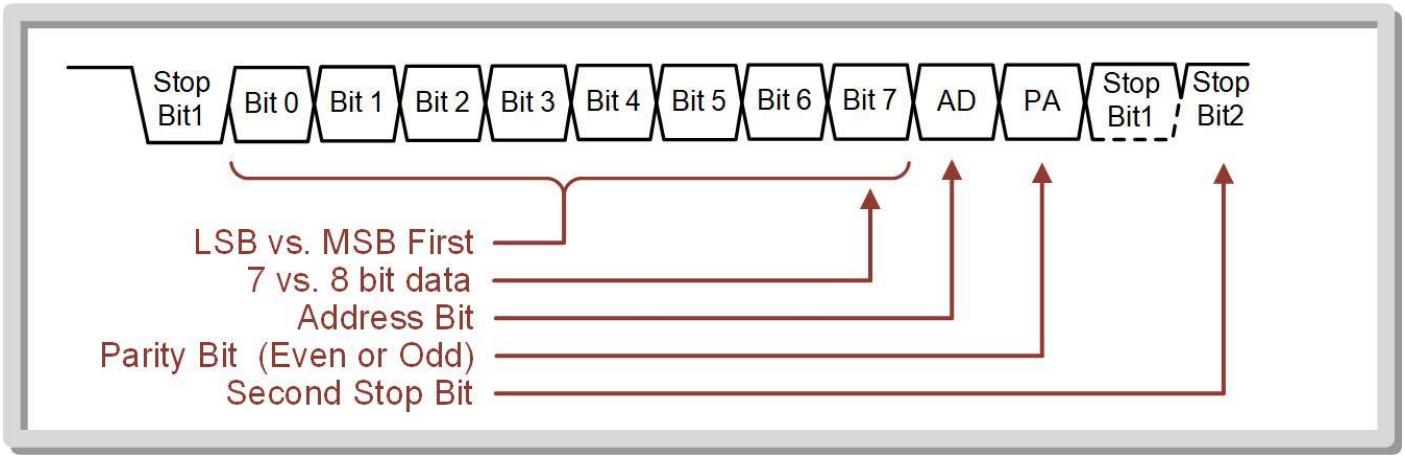
14.1.1 THE UART STANDARD

- The addition of a **second stop bit** can be used to provide extra time between packets when running at higher baud rates in order to make the link more reliable.



14.1.1 THE UART STANDARD

- **Parity bit** – used to detect transmission errors.
- **Even parity** – the transmitter counts the number of 1's in the data word and if the count is odd, it asserts the parity bit.
- **Odd parity** – the transmitter counts the number of 1's in the data word and if the count is even, it asserts the parity bit.



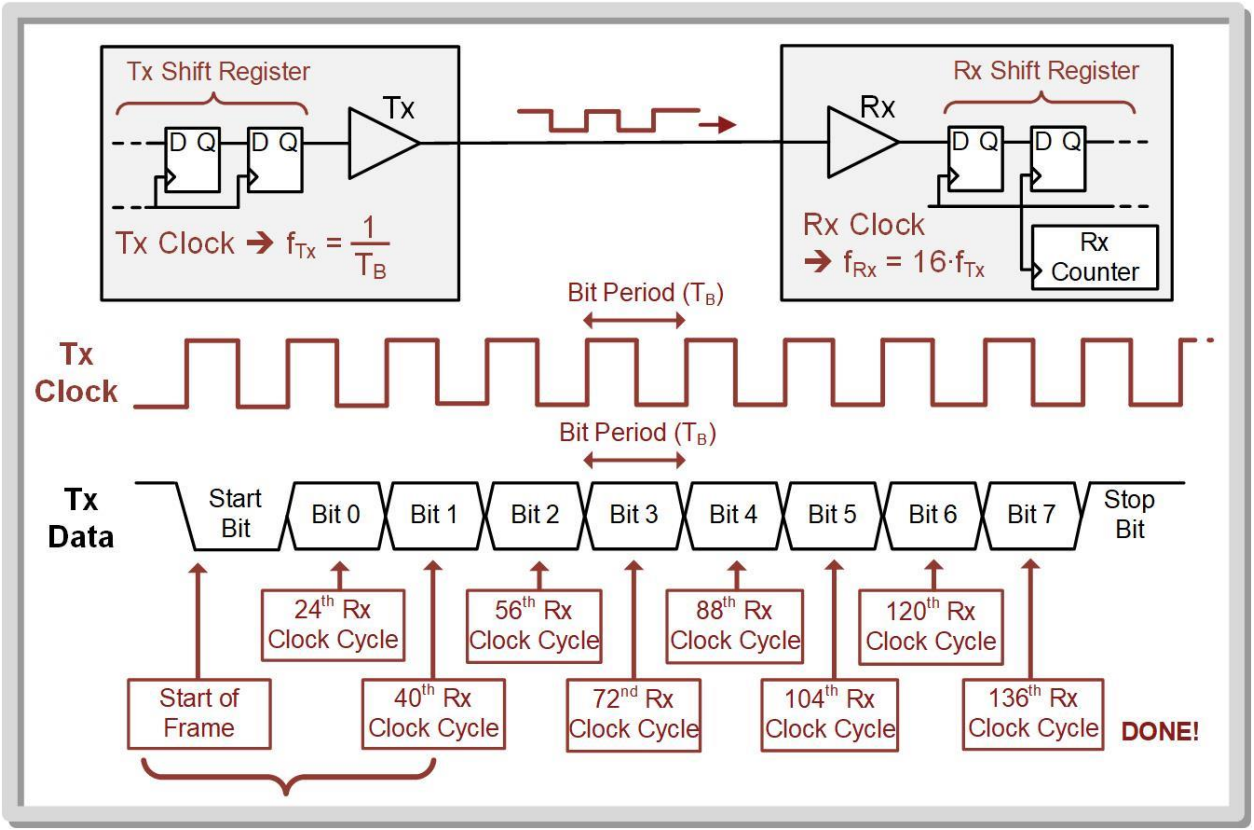
14.1.1 THE UART STANDARD

- **How does the receiver determine what logic is sent if it doesn't have a clock?**

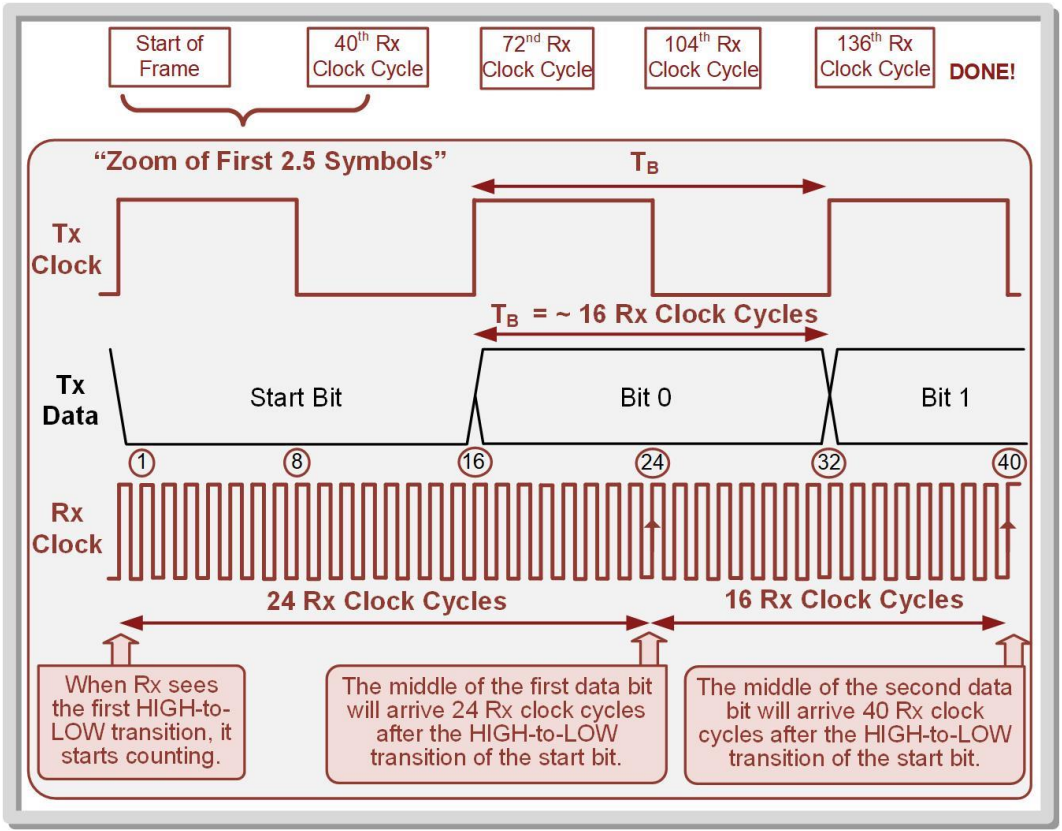
- **Oversampling** – using a receiver clock that is faster than the transmitter clock

Image Courtesy of https://neodem.wp.horizon.ac.uk/

14.1.1 THE UART STANDARD



14.1.1 THE UART STANDARD



14.1.1 THE UART STANDARD

- **Communication standard** – when logic levels are assigned to the UART logic values.
- As standard allows separate devices to communicate because both the logic levels and packet structure are known.
- The term transistor-to-transistor logic (TTL) is used to describe a UART that transmits a HIGH as the power supply (Vcc) of the device and a LOW as the GND of the device.
- **MSP430FR2355 Specifications:** Uses +3.4v TTL

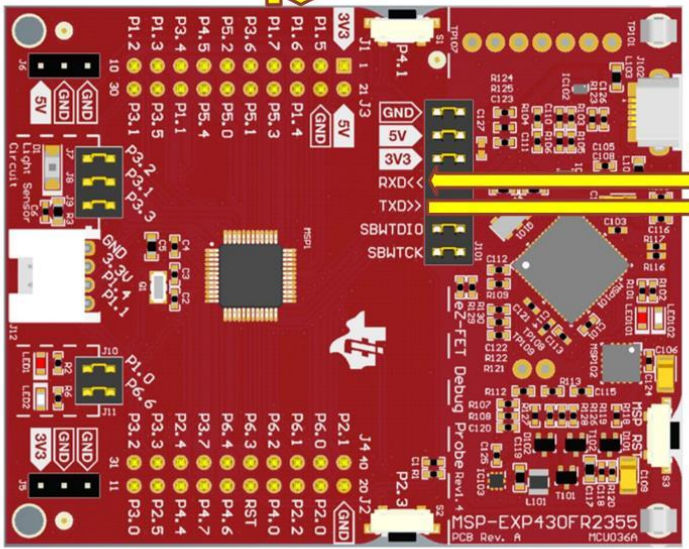
14.1.1 THE UART STANDARD

eUSCI_A0 (UART or SPI)
UART Tx (UCA0TXD) is on P1.7
UART Rx (UCA0RXD) is on P1.6

Configuring the P1 pins to be UART Tx/Rx is done in the **P1SEL1:P1SEL0** registers.

P1SEL1(7):P1SEL0(7) = b01 → P1.7 = UCA0TXD
P1SEL1(6):P1SEL0(6) = b01 → P1.6 = UCA0RXD

Tx Rx



Rx
Tx

eUSCI_A1 (UART or SPI)
UART Tx (UCA1TXD) is on P4.3
UART Rx (UCA1RXD) is on P4.2

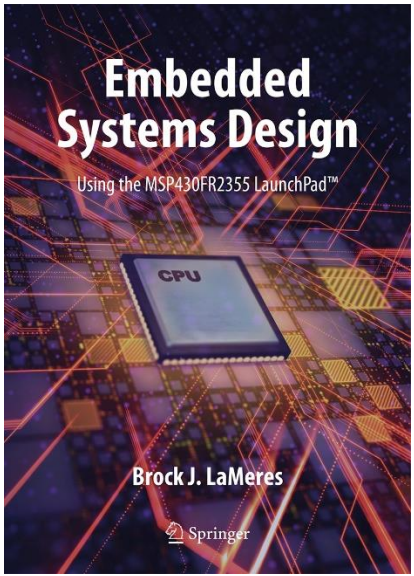
Configuring the P4 pins to be UART Tx/Rx is done in the **P4SEL1:P4SEL0** registers.

P4SEL1(2):P1SEL0(2) = b01
→ P4.2 = UCA1TXD

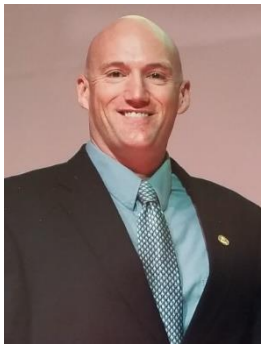
P4SEL1(3):P1SEL0(3) = b01
→ P4.3 = UCA1RXD

CHAPTER 14: SERIAL COMMUNICATIONS IN C

14.1.1 THE UART STANDARD

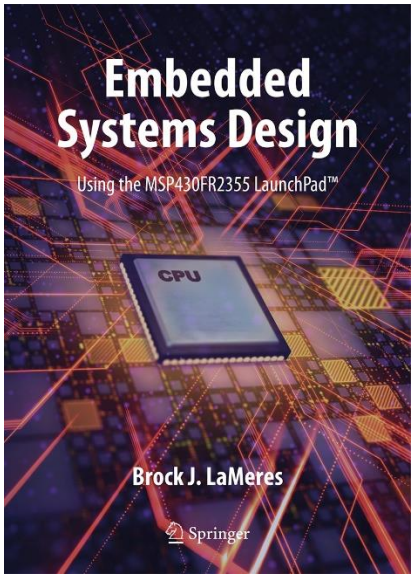


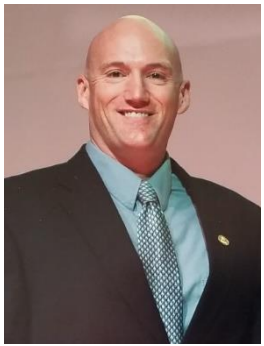
www.youtube.com/c/DigitalLogicProgramming_LaMeres



CHAPTER 14: SERIAL COMMUNICATIONS IN C

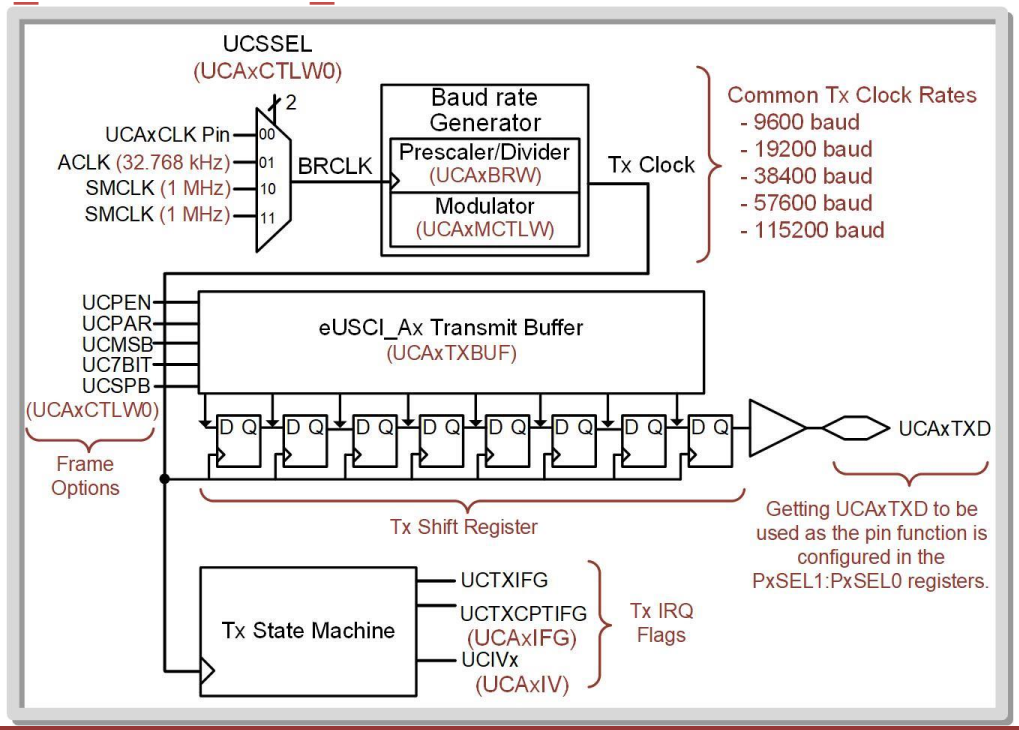
14.1.2 UART TRANSMIT ON THE MSP430FR2355 - CONFIGURATION





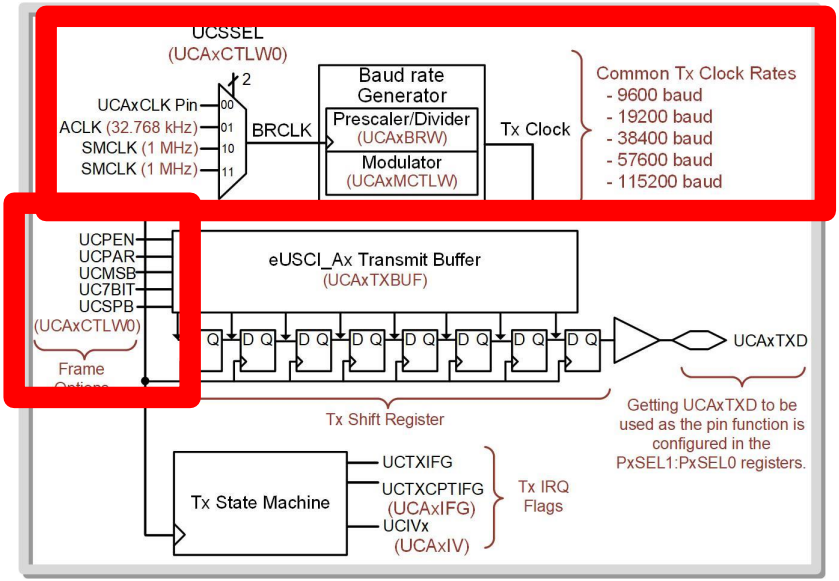
14.1.2 UART TRANSMIT ON THE MSP430FR2355

- The MSP430FR2355 contains two eUSCIs that support UARTs, eUSCI_A0 and eUSCI_A1.



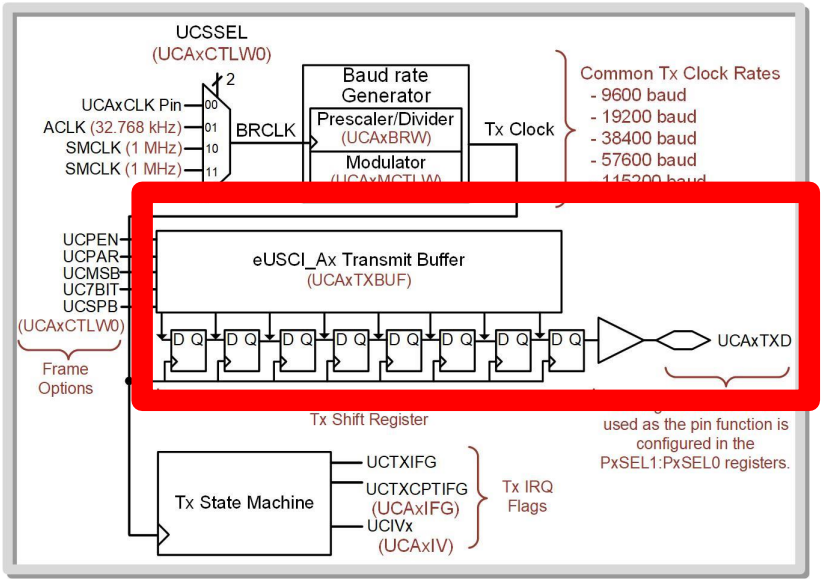
14.1.2 UART TRANSMIT ON THE MSP430FR2355

- The basic concept of operation of the UART system is to first configure its baud rate and frame characteristics.



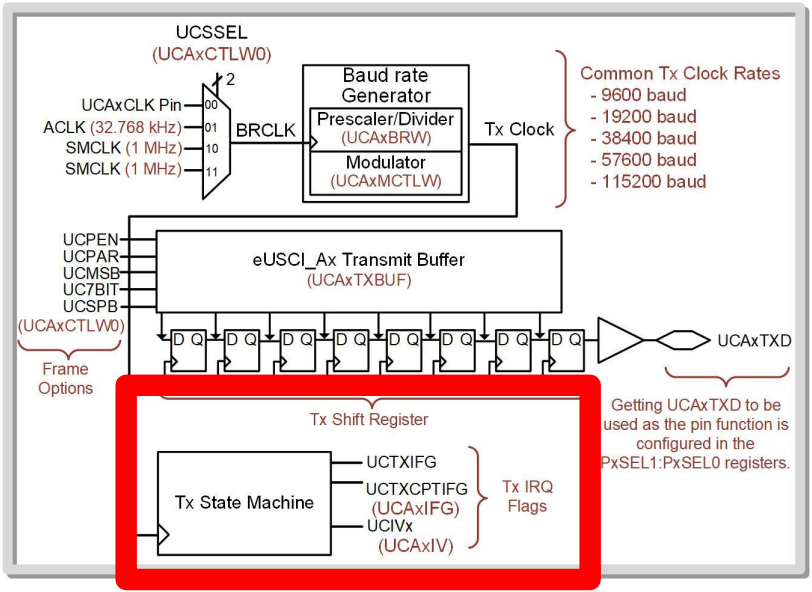
14.1.2 UART TRANSMIT ON THE MSP430FR2355

- We then store the data to be transmitted into a Tx buffer register, and a shift register will automatically send the data out over the Tx pin in a serial pattern.



14.1.2 UART TRANSMIT ON THE MSP430FR2355

- The UART can also produce a variety of interrupts such as **data received**, **transmit complete**, and many others used for detecting transmission errors.



14.1.2 UART TRANSMIT ON THE MSP430FR2355

Steps to Configure the UART Peripheral:

Step 1: Set the UCSWRST bit in the UCAxCTLW0 configuration register to put the eUSCI_Ax peripheral into reset. Note that the default value for UCSWRST = 1 on system power-on or system reset, so the system is initially in software reset.

Step 2: Initialize all eUSCI_Ax configuration registers.

Step 3: Configure ports.

Step 4: Clear UCSWRST to take the eUSCI_Ax peripheral out of reset.

Step 5: Enable interrupts (optional) using the UCRXIE or UCTXIE bits int eh UCAxIE configuration register.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- eUSCI_Ax Control Word 0 (UCAxCTLW0)
- eUSCI_Ax Control Word 1 (UCAxCTLW1)
- eUSCI_Ax Baud Rate Control Word (UCAxBRW)
- eUSCI_Ax Modulation Control Word (UCAxMCTLW)
- eUSCI_Ax Status (UCAxSTATW)
- eUSCI_Ax Receive Buffer (UCAxRXBUF)
- eUSCI_Ax Transmit Buffer (UCAxTXBUF)
- eUSCI_Ax Auto Baud Rate Control (UCAxABCTL)
- eUSCI_Ax IrDA Control (UCAxIRCTL)
- eUSCI_Ax Interrupt Enable (UCAxIE)
- eUSCI_Ax Interrupt Flag (UCAxIFG)
- eUSCI_Ax Interrupt Vector (UCAxIV)

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- We can start by just looking at the bare minimum to get the UART setup.

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- We can start by just looking at the bare minimum to get the UART setup.
- The UCAxCTLW0 register has a LOT of our settings.

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- We can start by just looking at the bare minimum to get the UART setup.
- The UCAxCTLW0 register has a LOT of our settings.
- First, let's consider what the default settings out of reset are:

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- We can start by just looking at the bare minimum to get the UART setup.
- The UCAxCTLW0 register has a LOT of our settings.
- First, let's consider what the default settings out of reset are:
  - We're by default in UART mode.

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- We can start by just looking at the bare minimum to get the UART setup.
- The UCAxCTLW0 register has a LOT of our settings.
- First, let's consider what the default settings out of reset are:
  - We also have standard framing options.

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- We can start by just looking at the bare minimum to get the UART setup.
- The UCAxCTLW0 register has a LOT of our settings.
- First, let's consider what the default settings out of reset are:
  - The UCSWRST bit is where we enable/disable the UART.
  - We need to put into Reset to configure the UART.
  - Note we "enable" a reset with UCSWRST=1

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- We can start by just looking at the bare minimum to get the UART setup.
- The UCAxCTLW0 register has a LOT of our settings.
- First, let's consider what the default settings out of reset are:
  - We can choose our clock source using the UCSSELx bits.
  - We usually use ACLK or SMCLK.

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- We can start by just looking at the bare minimum to get the UART setup.
- The UCAxCTLW0 register has a LOT of our settings.
- First, let's consider what the default settings out of reset are:
  - There are also options for error handling.
  - We won't be using these.

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

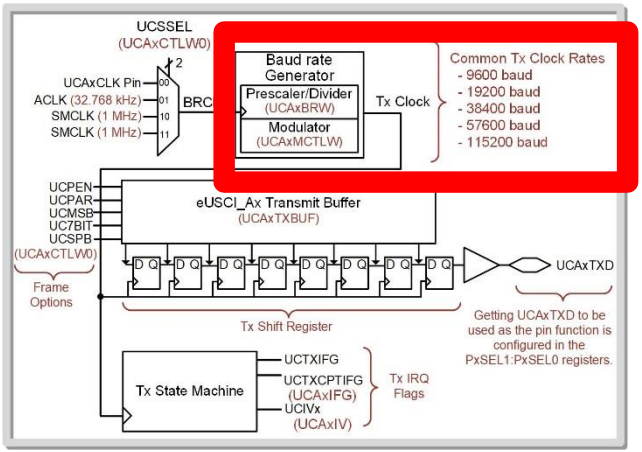
p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

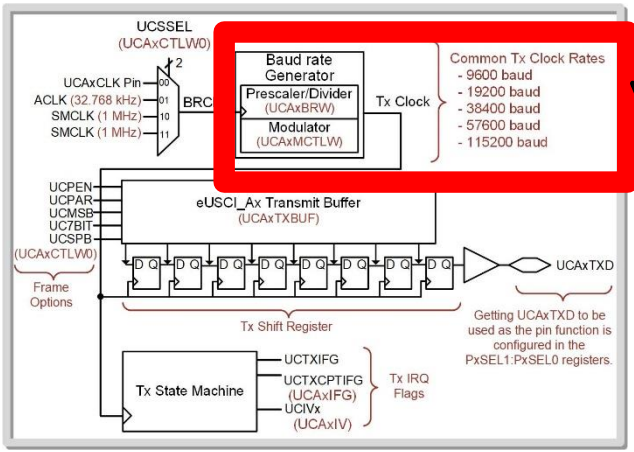
14.1.2 UART TRANSMIT ON THE MSP430FR2355

- The eUSCI_Ax peripheral has a baud rate generation circuit that can produce standard baud rates from nonstandard clock sources such as SMCLK (1MHz) and ACLK (32.768kHz).
- The baud rate is set using two configuration registers, the eUSCI_Ax Baud Rate Control Word (UCAxBRW) and eUSCI_Ax Modulation Control Word (UCAxMCTLW) regsiters.



14.1.2 UART TRANSMIT ON THE MSP430FR2355

- The eUSCI_Ax peripheral has a baud rate generation circuit that can produce standard baud rates from nonstandard clock sources such as SMCLK (1MHz) and ACLK (32.768kHz).
- The baud rate is set using two configuration registers, the eUSCI_Ax Baud Rate Control Word (UCAxBRW) and eUSCI_Ax Modulation Control Word (UCAxMCTLW) regsiters.



Note: This is complicated enough that I will put into a separate video.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- To configure the Tx and Rx ports, use the PxSEL1 and PxSEL0 configuration registers.

	PIN	UART	SPI	P1SEL1:0 Settings to Select eUSCI
eUSCI_A0	P1.7	Tx	SIMO	P1SEL1.7=0, P1SEL0.7 = 1
	P1.6	Rx	SOMI	P1SEL1.6=0, P1SEL0.6 = 1
	P1.5	-	SCLK	P1SEL1.5=0, P1SEL0.5 = 1
	P1.4	-	STE	P1SEL1.4=0, P1SEL0.4 = 1
	PIN	**UART**	**SPI**	**P4SEL1:0 Settings to Select eUSCI**
eUSCI_A1	P4.3	Tx	SIMO	P4SEL1.3=0, P4SEL0.3 = 1
	P4.2	Rx	SOMI	P4SEL1.2=0, P4SEL0.2 = 1
	P4.1	-	SCLK	P4SEL1.1=0, P4SEL0.1 = 1
	P4.0	-	STE	P4SEL1.0=0, P4SEL0.0 = 1
	PIN	**I2C**	**SPI**	**P1SEL1:0 Settings to Select eUSCI**
eUSCI_B0	P1.3	SCL	SIMO	P1SEL1.3=0, P1SEL0.3 = 1
	P1.2	SDA	SOMI	P1SEL1.2=0, P1SEL0.2 = 1
	P1.1	-	SCLK	P1SEL1.1=0, P1SEL0.1 = 1
	P1.0	-	STE	P1SEL1.0=0, P1SEL0.0 = 1
	PIN	**I2C**	**SPI**	**P4SEL1:0 Settings to Select eUSCI**
eUSCI_B1	P4.7	SCL	SIMO	P4SEL1.7=0, P4SEL0.7 = 1
	P4.6	SDA	SOMI	P4SEL1.6=0, P4SEL0.6 = 1
	P4.5	-	SCLK	P4SEL1.5=0, P4SEL0.5 = 1
	P4.4	-	STE	P4SEL1.4=0, P4SEL0.4 = 1

Configuring which eUSCI function to use within the peripheral is done using the UCSYNC bit in the **eUSCI_Ax Control Word Register 0** (UCAxCTLW0) or the UCMODEx bits in the **eUSCI_Bx Control Word Register 0** (UCBxCTLW0).

Configuring the pins to use the eUSCI capability is done using the **Port Function Select Registers** (PxSEL1:PPxSEL0).

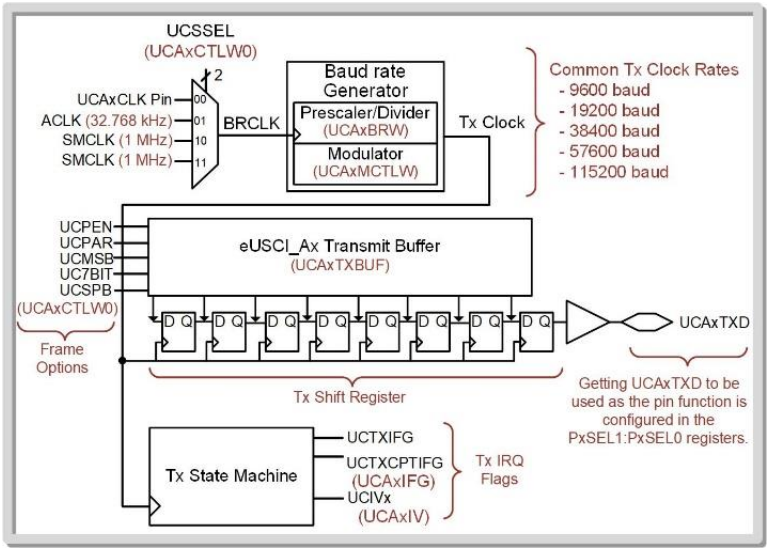
The default setting for PxSEL1:PxSEL0=00, which selects the digital I/O port function for each pin.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- Once taken out of software reset, the eUSCI_Ax module is enabled and will wait in an idle state until information is ready to be transmitted.

- A Tx is initiated by writing a byte to the eUSCI_Ax Transmit Buffer (UCAxTXBUF).

- The data in UCAxTXBUF is moved into the Tx shift
- register and the baud rate generator beings producing clocks.



14.1.2 UART TRANSMIT ON THE MSP430FR2355

- The UCTXIFG (in UCAxIFG) flag provides the status of the transmission.
- When UCTXIFG = 0, data is being shifted out and new data should not be written to the Tx buffer.
- When UCTXIFG = 1, new data can be written to the Tx buffer

eUSCI_Ax Transmit Buffer (UCAxTXBUF) Register

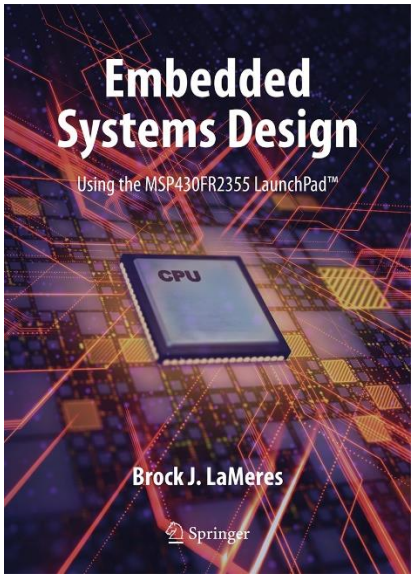
p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								UCTXBUFx							
Value on Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:8	Reserved	-
7:0	UCTXBUFx	This register holds the data waiting to be moved into the transmit shift register. Once this value is moved into the shift register it will immediately begin being shifted out. Writing to this register clears the UCTXIFG flag.

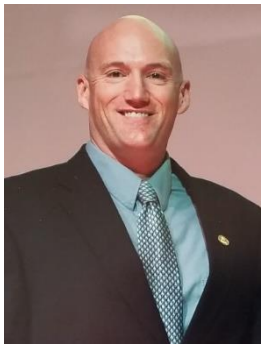
Embedded Systems Design

Chapter 14: Serial Communications in C

14.1.2 UART Transmit on the MSP430FR2355 - Configuration



www.youtube.com/c/DigitalLogicProgramming_LaMeres

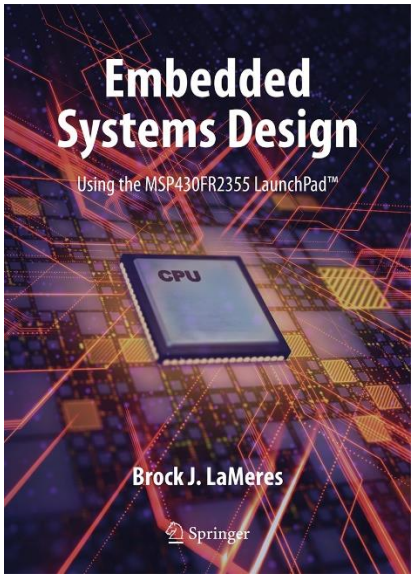


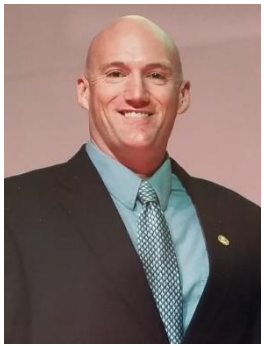
Brock J. LaMeres, Ph.D.

Embedded Systems Design

Chapter 14: Serial Communications in C

14.1.2 UART Transmit on the MSP430FR2355 – Setting Baud Rate

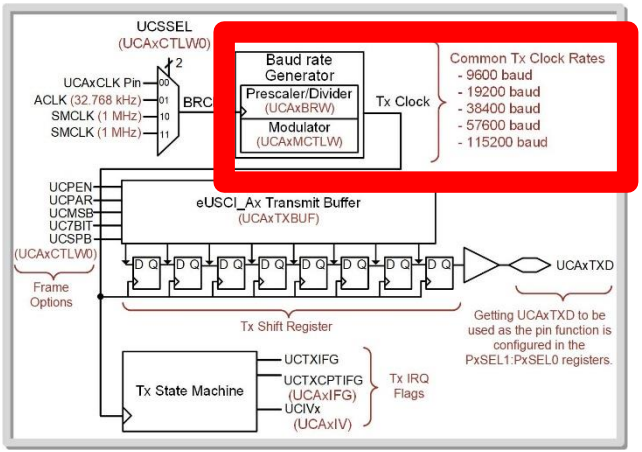




Brock J. LaMeres, Ph.D.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- The eUSCI_Ax peripheral has a baud rate generation circuit that can produce standard baud rates from nonstandard clock sources such as SMCLK (1MHz) and ACLK (32.768kHz).
- The baud rate is set using two configuration registers, the eUSCI_Ax Baud Rate Control Word (UCAxBRW) and eUSCI_Ax Modulation Control Word (UCAxMCTLW) regsiters.



14.1.2 UART TRANSMIT ON THE MSP430FR2355

eUSCI_Ax Baud Rate Control Word (UCAxBRW) Register

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	UCAxBRW															
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
Reset:																

Bit	Field	Description
15:0	UCAxBRW	Clock Prescalar Settings of the Baud Rate Generator

Note: Only modify when UCSWRST=1.

EX: CALCULATING ERROR OF EUSCI PRESCALER DEVISION

Calculate how much error is present in the eUSCI UART baud rate if we only use a prescaler to divide down the source clock. Assume f_{BRCLK}=1 MHz and the desired baud rate is 115,200.

First, let's find the prescalar value that will get us as close as possible to the desired baud rate.

$$N = \frac{f_{BRCLK}}{\text{desired baud rate}} = \frac{1\ \text{MHz}}{115{,}200\ \text{baud}} = 8.668$$

We can only place the integer portion of this number (i.e., 8) into the UCAxBRW register to set the number of times the source clock is divided. This is called the "prescalar" value. Let's see what the resulting baud rate is if only a prescalar divider is used.

$$\text{prescaler baud rate} = \frac{f_{BRCLK}}{N} = \frac{1\ \text{MHz}}{8} = 125{,}000\ \text{baud}$$

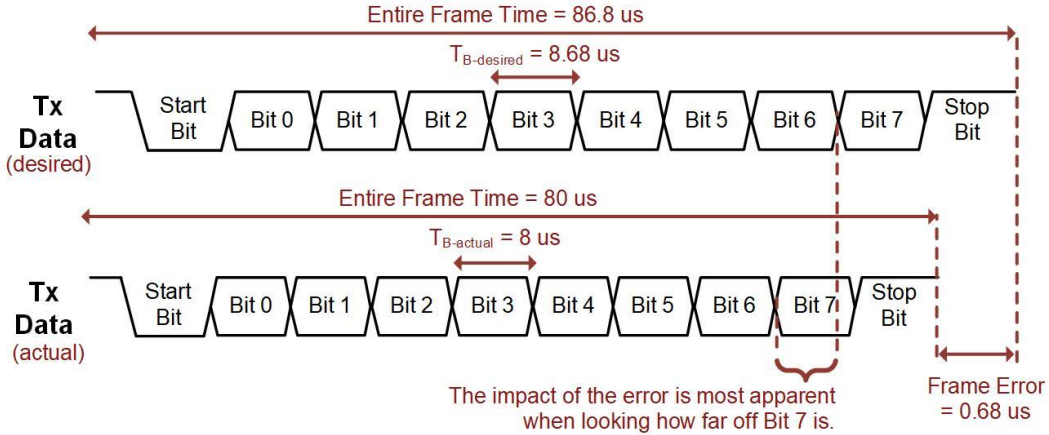
This result is not exactly our desired baud rate of 115,200. Let's calculate how much error is present in our baud rate if we use the value produced by only the prescaler.

$$\%_{error} = \frac{|\#_{actual} - \#_{desired}|}{\#_{desired}} \times 100 = \frac{|125{,}000 - 115{,}200|}{115{,}200} \times 100 = 8.5\%$$

EX: CALCULATING ERROR OF EUSCI PRESCALER DEVISION

This error causes the baud rate being transmitted to be significantly different from the timing that the Rx is expecting.

$$T_{B\text{-desired}} = \frac{1}{115{,}200} = 8.68 \text{ us} \qquad T_{B\text{-actual}} = \frac{1}{125{,}000} = 8 \text{ us}$$



14.1.2 UART TRANSMIT ON THE MSP430FR2355

- Modulation circuits dynamically add and subtract BRCLK periods to the baud rate clock that drives the shift register.

- By adding small amounts of time from the BRCLK clock period to the baud rate clock, the timing error can be reduced.

- In low frequency mode, the baud rate generator uses the second modulation stage to create the desired baud rate.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

eUSCI_Ax Modulation Control Word (UCAxMCTLW) Register

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	UCBRSx								UCBRFx				Reserved			UCOS16
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:8	UCBRSx	Second Modulation Stage Select. These bits hold a free modulation pattern for BITCLK.
7:4	UCBRFx	First Modulation Stage Select. These bits determine the modulation pattern for BITCLK16 when UCOS16=1. Ignored when UCOS16=0. the “Oversampling Baud-Rate Generation” section of the MSP430 User’s Guide shows the modulation pattern.
3:1	Reserved	Reserved
0	UCOS16	Oversampling Mode Enable (0=disabled; 1=enabled)

Note: Only modify when UCSWRST=1.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- **Low frequency baud rate generation** – used for baud rates that are less than or equal to 1/3 of BRCLK while the oversampling baud rate generation is used for higher ratios.

- The UCSO16 bit in the UCAxMCTLW register controls the modulation mode with UCSO16 = 0 being for low frequency (default) and UCSO16 = 1 being for oversampling mode.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- In oversampling baud rate mode, both the first and second modulation stages are used to reduce this timing error.

- The first modulation stage is configured using the UCBRFx bits within the UCAxMCTLW register.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

BRCLK	f_{BRCLK}	Baud Rate	UCBRx	UCOS16	UCBRFx	UCBRSx
ACLK (UCSSEL=01)	32.768 kHz	1200	1	1	11	0x25
	32.768 kHz	2400	13	0	-	0xB6
	32.768 kHz	4800	6	0	-	0xEE
	32.768 kHz	9600	3	0	-	0x92
SMCLK (UCSSEL=10 or UCSSEL=11)	1 MHz	9600	6	1	8	0x20
	1 MHz	19200	3	1	4	0x02
	1 MHz	38400	1	1	10	0x00
	1 MHz	57600	17	0	-	0x4A
	1 MHz	115200	8	0	-	0xD6

UCSSEL is configured in the **UCAxCTLW0** register.

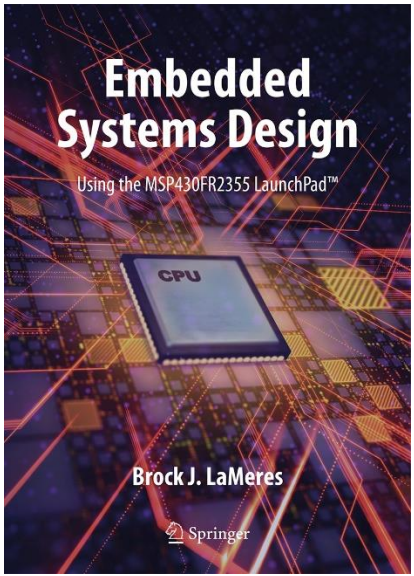
This value is stored in the **UCAxBRW** register.

These bits are configured in the **UCAxMCTL** register.

Embedded Systems Design

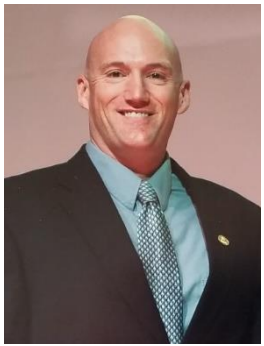
Chapter 14: Serial Communications in C

14.1.2 UART Transmit on the MSP430FR2355 – Setting Baud Rate



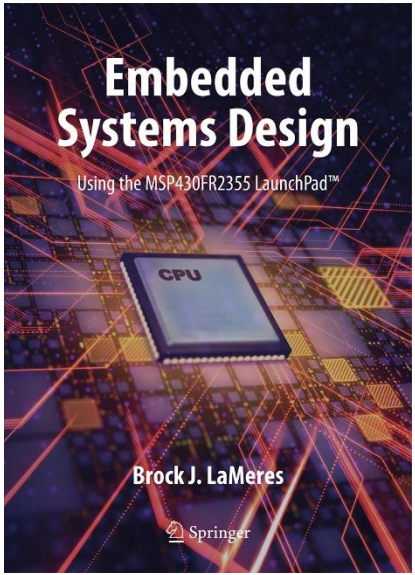
Subscribe

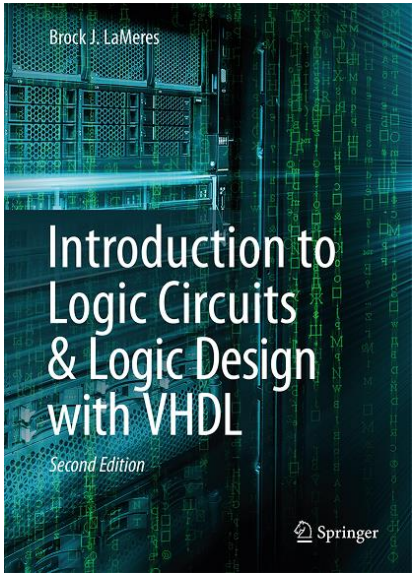
www.youtube.com/c/DigitalLogicProgramming_LaMeres



The Analog Discovery 2

Introduction to the AD2 + Installing Waveforms

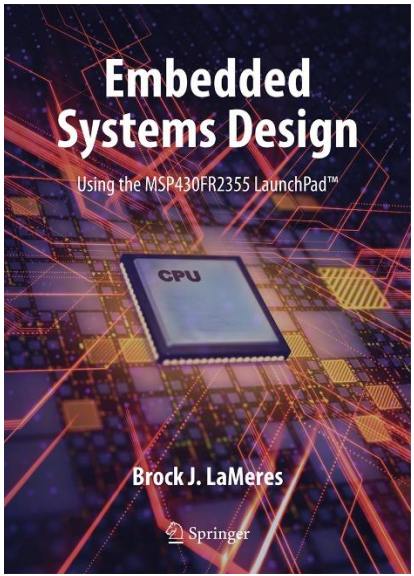


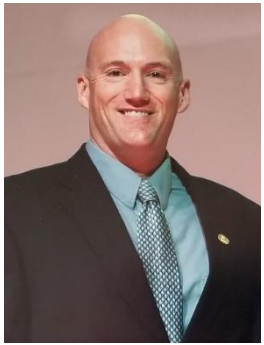


Brock J. LaMeres, Ph.D.

CHAPTER 14: SERIAL COMMUNICATIONS IN C

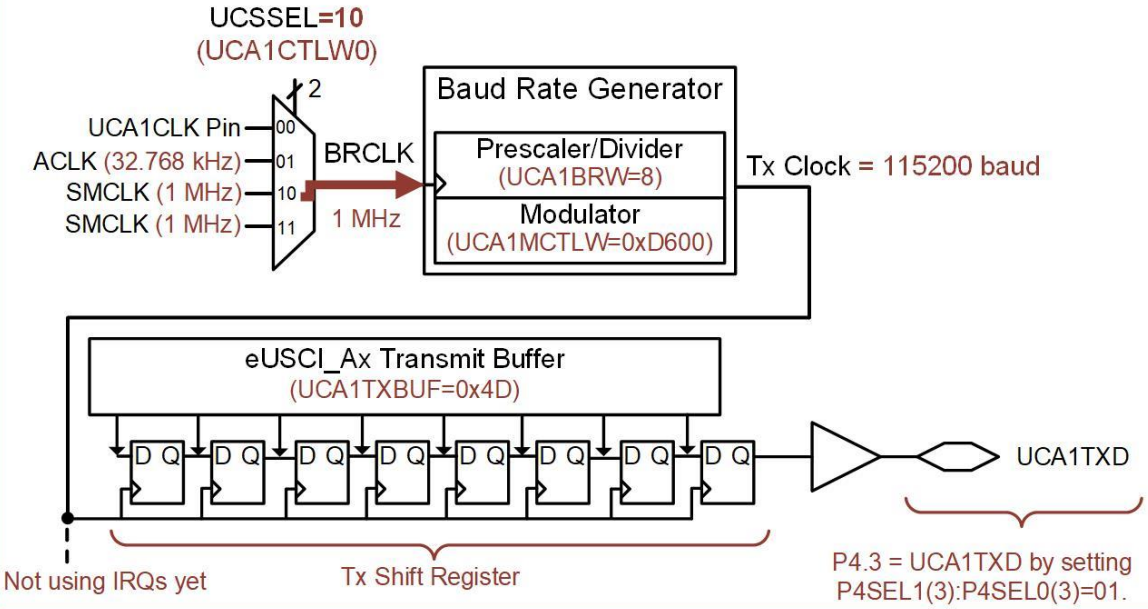
14.1.2 UART TRANSMIT ON THE MSP430FR2355 – EXAMPLE: TRANSMITTING A BYTE OVER UART AT 115,200 BAUD





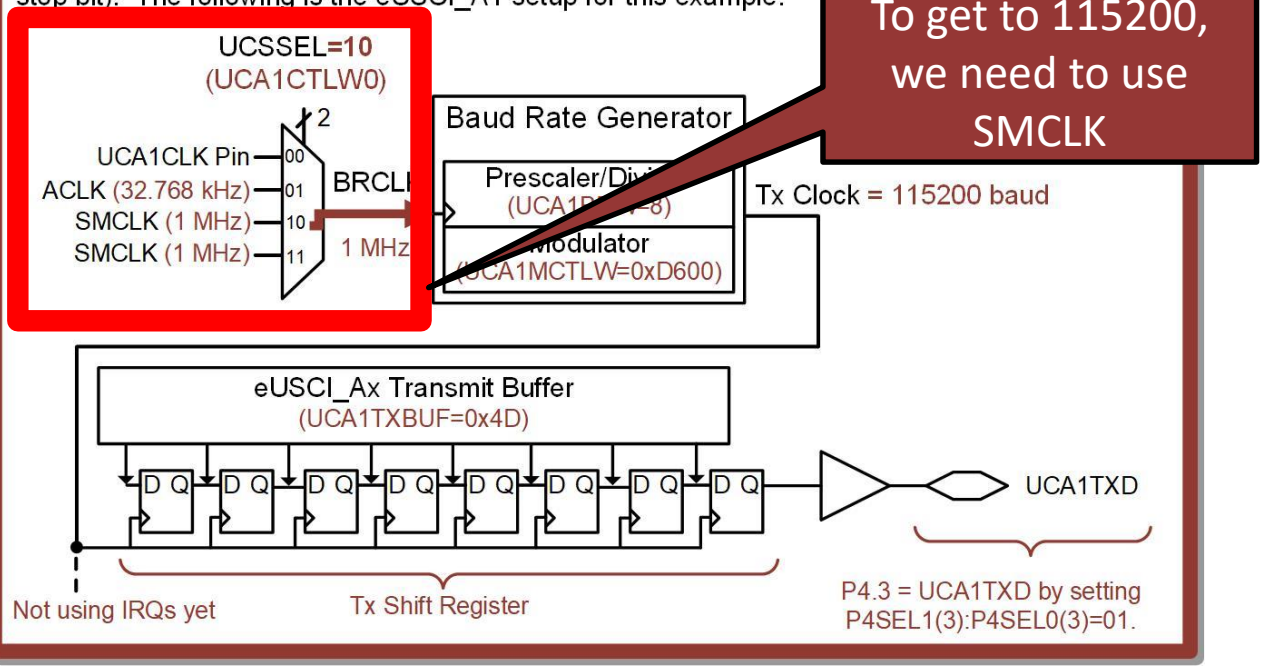
EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

Let's configure the eUSCI_A1 peripheral as a UART and transmit the value **0x4D** at a rate of 115200 baud. The Tx output for eUSCI_A1 is on P4.3. We will use SMCLK as the source clock (BRCLK) and the normal UART framing options (8-bit, LSB first, no parity, no address, no extra stop bit). The following is the eUSCI_A1 setup for this example.



EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

Let's configure the eUSCI_A1 peripheral as a UART and transmit the value **0x4D** at a rate of 115200 baud. The Tx output for eUSCI_A1 is on P4.3. We will use SMCLK as the source clock (BRCLK) and the normal UART framing options (8-bit, LSB first, no parity, no address, no extra stop bit). The following is the eUSCI_A1 setup for this example.



EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

BRCLK	f_{BRCLK}	Baud Rate	UCBRx	UCOS16	UCBRFx	UCBRSx
ACLK (UCSSEL=01)	32.768 kHz	1200	1	1	11	0x25
	32.768 kHz	2400	13	0	-	0xB6
	32.768 kHz	4800	6	0	-	0xEE
	32.768 kHz	9600	3	0	-	0x92
SMCLK (UCSSEL=10 or UCSSEL=11)	1 MHz	9600	6	1	8	0x20
	1 MHz	19200	3	1	4	0x02
	1 MHz	38400	1	1	10	0x00
	1 MHz	115200	8	0	-	0xD6

UCSSEL is configured in the **UCAxCTLW0** register.

This value is stored in the **UCAxBRW** register.

These bits are configured in the **UCAxMCTL** register.

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

BRCLK	f_{BRCLK}	Baud Rate	UCBRx	UCOS16	UCBRFx	UCBRSx
ACLK (UCSSEL=01)	32.768 kHz	1200	1	1	11	0x25
	32.768 kHz	2400	13	0	-	0xB6
	32.768 kHz	4800	6	0	-	0xEE
	32.768 kHz	9600	3	0	-	0x92
SMCLK (UCSSEL=10 or UCSSEL=11)	1 MHz	9600	6	1	8	0x20
	1 MHz	19200	3	1	4	0x02
	1 MHz	38400	1	1	10	0x00
	1 MHz	57600	17	0	-	0x4A
	1 MHz	115200	8	0	-	0xD6

UCSSEL is configured in the **UCAxCTLW0** register.

This value is stored in the **UCAxBRW** register.

These bits are configured in the **UCAxMCTL** register.

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

eUSCI_Ax Baud Rate Control Word (UCAxBRW) Register

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	UCAxBRW															
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
Reset:																

Bit	Field	Description
15:0	UCAxBRW	Clock Prescalar Settings of the Baud Rate Generator

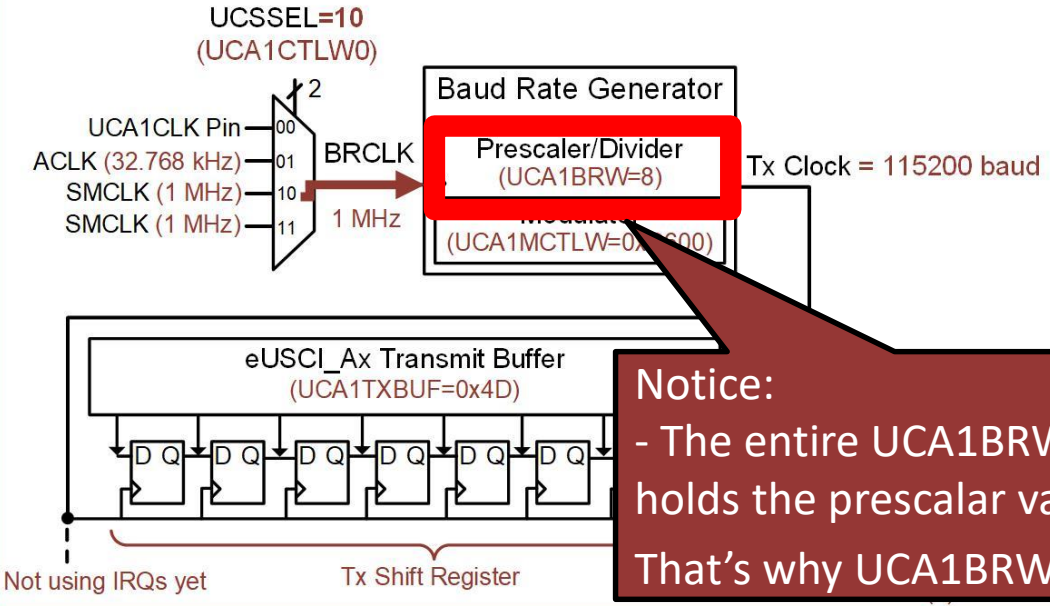
Note: Only modify when UCSWRST=1.

Notice:
- The entire UCA1BRW register holds the prescalar value.

That's why UCA1BRW = 8;

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

Let's configure the eUSCI_A1 peripheral as a UART and transmit the value **0x4D** at a rate of 115200 baud. The Tx output for eUSCI_A1 is on P4.3. We will use SMCLK as the source clock (BRCLK) and the normal UART framing options (8-bit, LSB first, no parity, no address, no extra stop bit). The following is the eUSCI_A1 setup for this example.



Notice:
- The entire UCA1BRW register holds the prescalar value.

That's why UCA1BRW = 8;

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

BRCLK	f_{BRCLK}	Baud Rate	UCBRx	UCOS16	UCBRFx	UCBRSx
ACLK (UCSSEL=01)	32.768 kHz	1200	1	1	11	0x25
	32.768 kHz	2400	13	0	-	0xB6
	32.768 kHz	4800	6	0	-	0xEE
	32.768 kHz	9600	3	0	-	0x92
SMCLK (UCSSEL=10 or UCSSEL=11)	1 MHz	9600	6	1	8	0x20
	1 MHz	19200	3	1	4	0x02
	1 MHz	38400	1	1	10	0x00
	1 MHz	57600	17	0		
	1 MHz	115200	8	0	-	0xD6

UCSSEL is configured in the **UCAxCTLW0** register.

This value is stored in the **UCAxBRW** register.

These bits are configured in the **UCAxMCTL** register.

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

eUSCI_Ax Modulation Control Word (UCAxMCTLW) Register

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	UCBRSx								UCBRFx				Reserved			UCOS16
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:8	UCBRSx	Second Modulation Stage Select. These bits hold a free modulation for BITCLK.
7:4	UCBRFx	First Modulation Stage Select. These bits determine the modulati BITCLK16 when UCOS16=1. Ignored when UCOS16=0. the "Ov Baud-Rate Generation" section of the MSP430 User's Guide show modulation pattern.
3:1	Reserved	Reserved
0	UCOS16	Oversampling Mode Enable (0=disabled; 1=enabled)

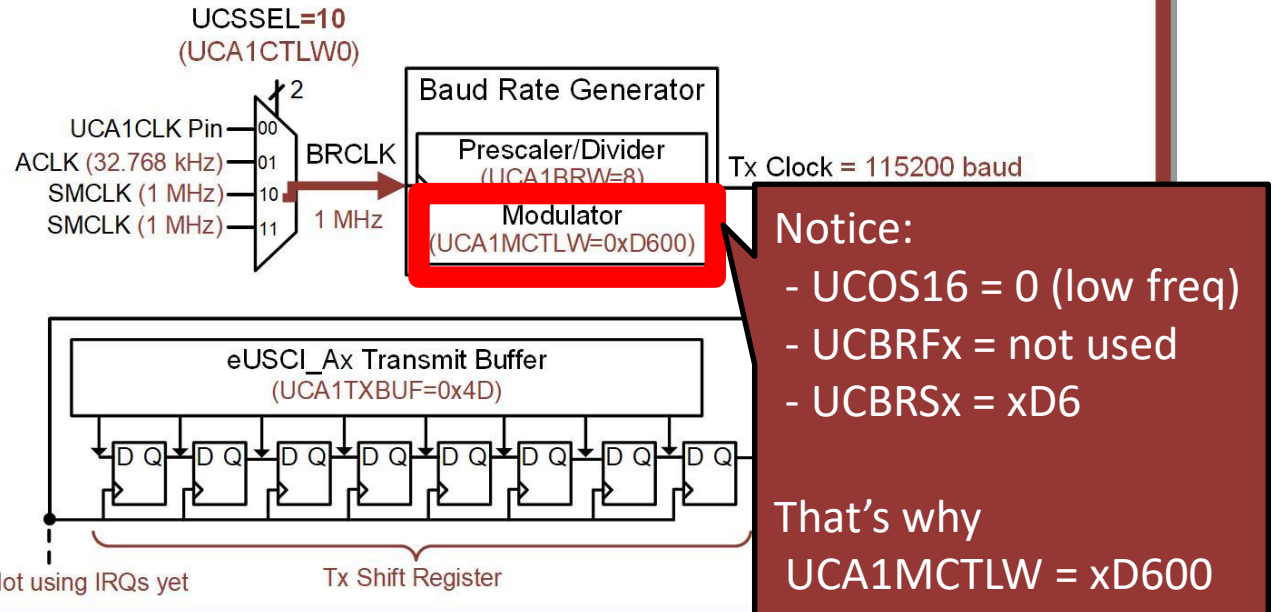
Note: Only modify when UCSWRST=1.

Notice:
- UCOS16 = 0 (low freq)
- UCBRFx = not used
- UCBRSx = xD6

That's why
UCA1MCTLW = xD600

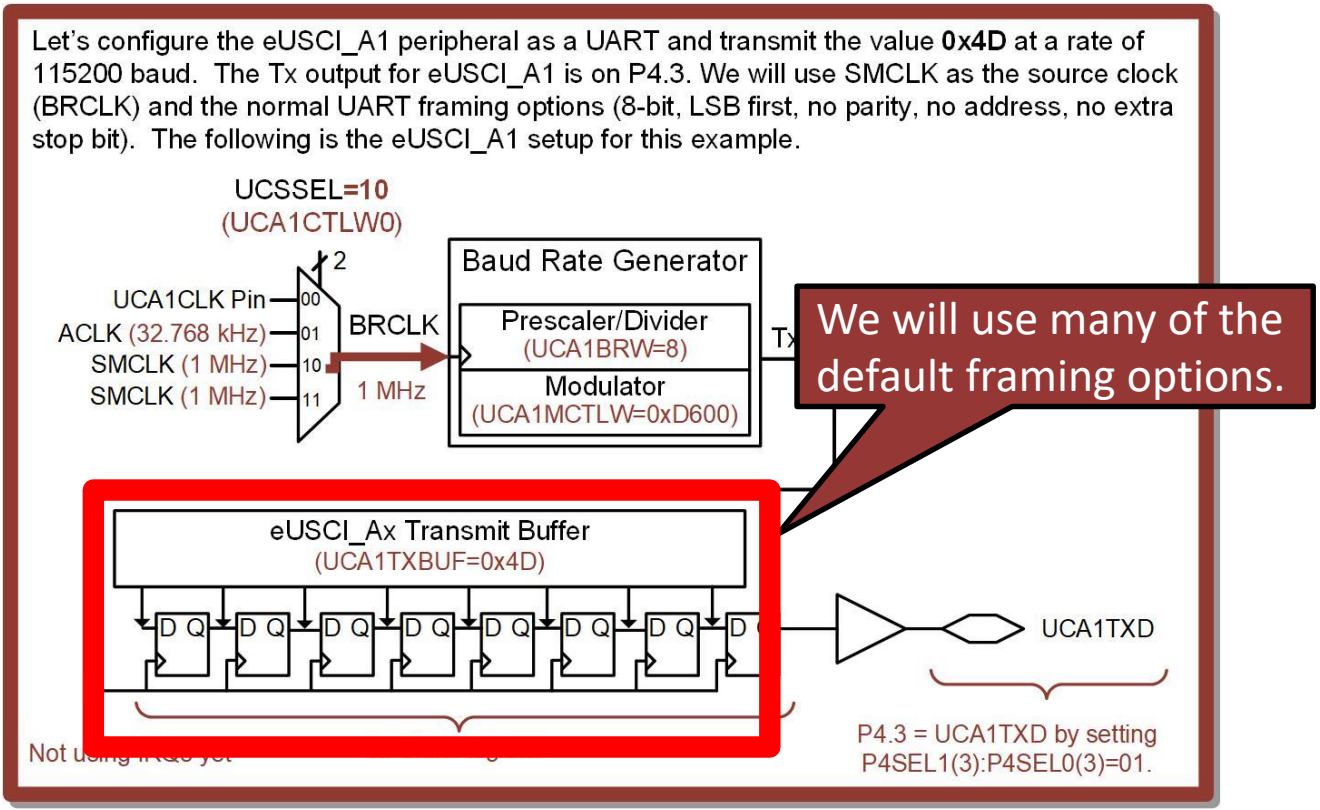
EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

Let's configure the eUSCI_A1 peripheral as a UART and transmit the value **0x4D** at a rate of 115200 baud. The Tx output for eUSCI_A1 is on P4.3. We will use SMCLK as the source clock (BRCLK) and the normal UART framing options (8-bit, LSB first, no parity, no address, no extra stop bit). The following is the eUSCI_A1 setup for this example.



EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

Let's configure the eUSCI_A1 peripheral as a UART and transmit the value **0x4D** at a rate of 115200 baud. The Tx output for eUSCI_A1 is on P4.3. We will use SMCLK as the source clock (BRCLK) and the normal UART framing options (8-bit, LSB first, no parity, no address, no extra stop bit). The following is the eUSCI_A1 setup for this example.



EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

The settings out of reset give us:

- No parity
- LSB First
- 8-Bit Length
- 1 Stop Bit

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

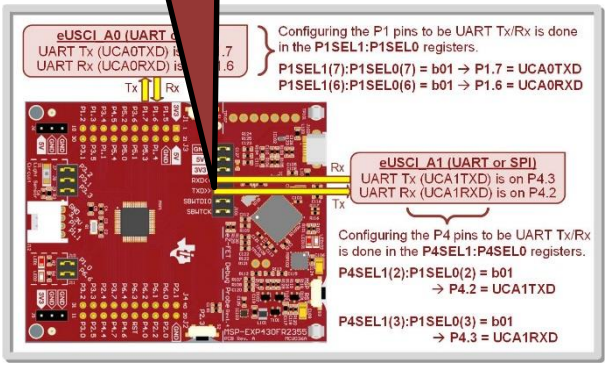
	PIN	UART	SPI	P1SEL1:0 Settings to Select eUSCI
eUSCI_A0	P1.7	Tx	SIMO	P1SEL1.7=0, P1SEL0.7 = 1
	P1.6	Rx	SOMI	P1SEL1.6=0, P1SEL0.6 = 1
	P1.5	-	SCLK	P1SEL1.5=0, P1SEL0.5 = 1
	P1.4	-	STE	P1SEL1.4=0, P1SEL0.4 = 1
eUSCI_A1	P4.3	Tx	SIMO	P4SEL1.3=0, P4SEL0.3 = 1
	P4.1	-	SCLK	P4SEL1.1=0, P4SEL0.1 = 1
	P4.0	-	STE	P4SEL1.0=0, P4SEL0.0 = 1
	PIN	**I2C**	**SPI**	**P1SEL1:0 Settings to Select eUSCI**
eUSCI_B0	P1.3	SCL	SIMO	P1SEL1.3=0, P1SEL0.3 = 1
	P1.2	SDA	SOMI	P1SEL1.2=0, P1SEL0.2 = 1
	P1.1	-	SCLK	P1SEL1.1=0, P1SEL0.1 = 1
	P1.0	-	STE	P1SEL1.0=0, P1SEL0.0 = 1
	PIN	**I2C**	**SPI**	**P4SEL1:0 Settings to Select eUSCI**
eUSCI_B1	P4.7	SCL	SIMO	P4SEL1.7=0, P4SEL0.7 = 1
	P4.6	SDA	SOMI	P4SEL1.6=0, P4SEL0.6 = 1
	P4.5	-	SCLK	P4SEL1.5=0, P4SEL0.5 = 1
	P4.4	-	STE	P4SEL1.4=0, P4SEL0.4 = 1

Configuring which eUSCI function to use within the peripheral is done using the UCSYNC bit in the **eUSCI_Ax Control Word Register 0** (UCAxCTLW0) or the UCMODEx bits in the **eUSCI_Bx Control Word Register 0** (UCBxCTLW0).

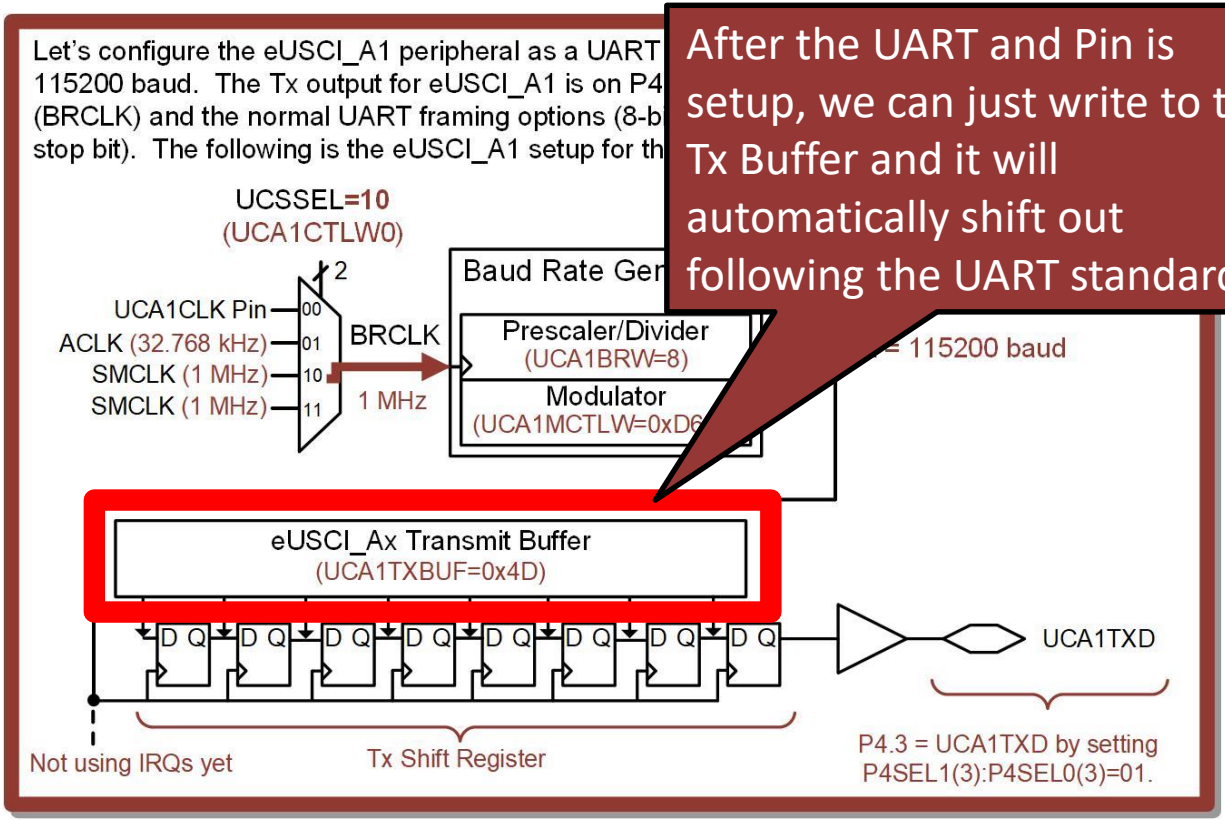
Configuring the pins to use the eUSCI capability is done using the **Port Function Select Registers** (PxSEL1:PPxSEL0).

The default setting for PxSEL1:PxSEL0=00, which selects the digital I/O port function for each pin.

This pin is brought out to a pin on J101



EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD



EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

Step 1: In CCS, create a new C/C++ Empty Project (with main.c) titled:

C_UART_Tx1_Sending_Byte_at_115200

Step 2: Type in the following code in main.c after the statement to stop the watchdog timer.

Image Courtesy of https://neodem.wp.horizon.ac.uk/

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

```
#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;

    //-- 1. Put eUSCI_A1 into SW reset
    UCA1CTLW0 |= UCSWRST;

    //-- 2. Configure eUSCI_A1
    UCA1CTLW0 |= UCSSEL__SMCLK;

    UCA1BRW = 8;
    UCA1MCTLW |= 0xD600;

    //-- 3. Configure Ports
    P4SEL1 &= ~BIT3;
    P4SEL0 |= BIT3;

    PM5CTL0 &= ~LOCKLPM5;

    //-- 4. Take eUSCI_A1 out of software reset
    UCA1CTLW0 &= ~UCSWRST;

  int i;

  while(1)
  {
      UCA1TXBUF = 0x4D;
      for(i=0; i<10000; i=i+1){}
  }

  return 0;
}
```

Put eUSCI_A1 into SW reset with UCSWRST=1

Choose SMCLK=1MHz as BRCLK

N=1M/115200 = 8.68 → "Low Frequency Baud Rate Mode"
Prescaler → UCA1_BRW = 8
Modulation → UCA1MCTLW = 0xD600 (from table)

Configure P4.3 to use UCA1TXD with:
P4SEL1(3):P3SEL0(3)=01

Take eUSCI_A1 out of SW reset with UCSWRST=0

Put 0x4D in transmit buffer. As soon as it is stored, it will begin shifting out the data.

In order to not overwrite the buffer before it shifts all of the bits out, we will insert a delay.

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

Step 3: Debug your program and run it.

Note: We will first look at the Tx signal with an oscilloscope. If you don't have an oscilloscope, we will look at the signal in a different way in a later example.

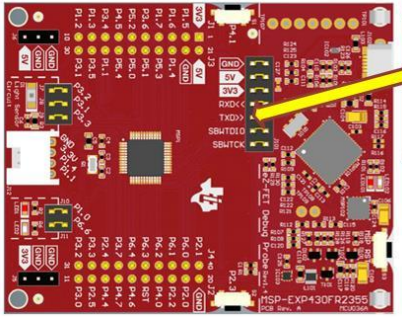
Image Courtesy of https://neodem.wp.horizon.ac.uk/

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD

Step 4: Observe the UART with an oscilloscope. Remove the jumper on the pins labeled “TXD>>” on the J101 header of the MSP430FR2355 board. Probe the pin nearest the MSP430 MCU (MSP1). This pin is being driven by UCA1TXD.

Step 5: Configure the oscilloscope waveform to center the UART frame. Measure the bit period of one of the short pulses.

EX: TRANSMITTING A BYTE OVER UART AT 115200 BAUD



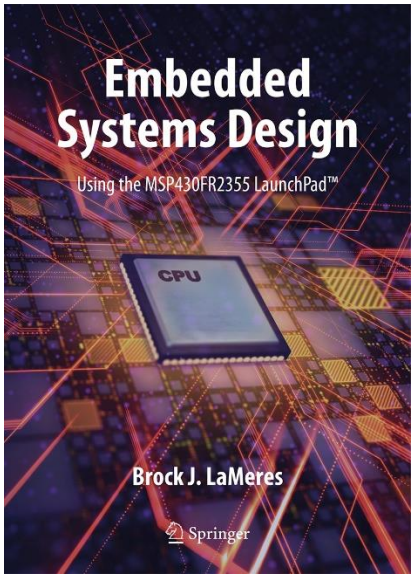
Remove this jumper and probe with an oscilloscope.

The bit period is ~8.6us, which corresponds to a ~115200 baud rate.

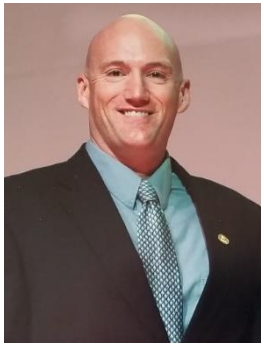
Did it work? Are you able to see the UART frame at 115200 baud? Notice that the sequence of bits received was 0b10110010. But remember that the UART sends the LSB first. Putting this in order from MSB we get: 0b01001101=0x4D.

CHAPTER 14: SERIAL COMMUNICATIONS IN C

14.1.2 UART TRANSMIT ON THE MSP430FR2355 – EXAMPLE: TRANSMITTING A BYTE OVER UART AT 115,200 BAUD

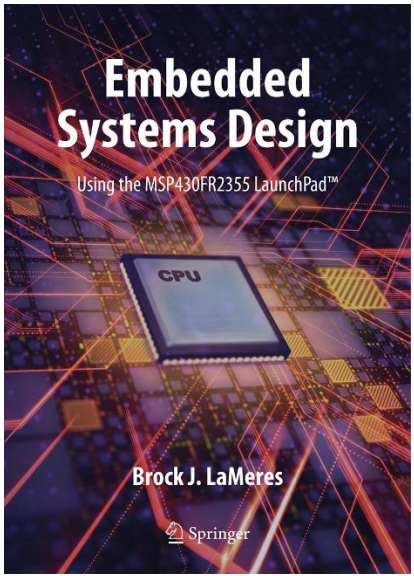


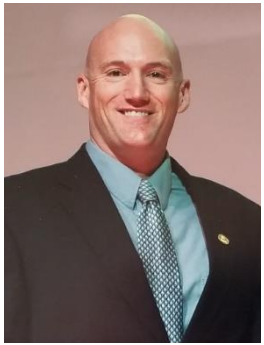
www.youtube.com/c/DigitalLogicProgramming_LaMeres



CHAPTER 14: SERIAL COMMUNICATIONS IN C

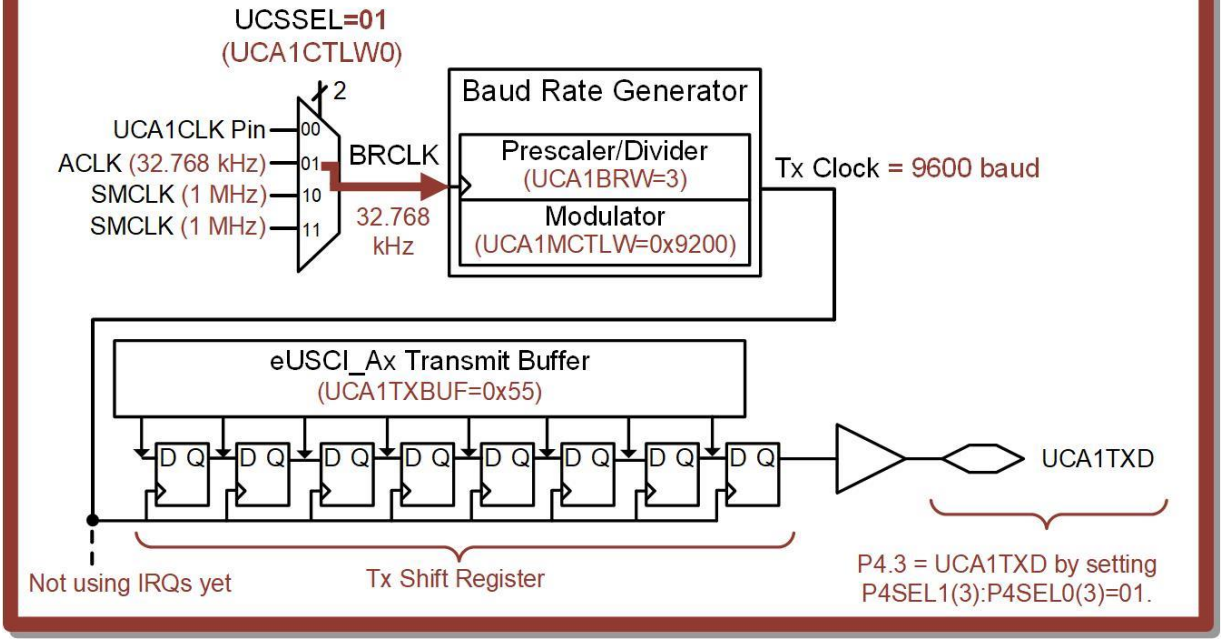
14.1.2 UART TRANSMIT ON THE MSP430FR2355 – EXAMPLE: TRANSMITTING A BYTE OVER UART AT 9600 BAUD





EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

Let's configure the eUSCI_A1 peripheral as a UART and transmit the value **0x55** at a rate of 9600 baud. The Tx output for eUSCI_A1 is on P4.3. We will use ACLK as the source clock (BRCLK) and the normal UART framing options (8-bit, LSB first, no parity, no address, no extra stop bit). The following is the eUSCI_A1 setup for this example.



EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

BRCLK	f_{BRCLK}	Baud Rate	UCBRx	UCOS16	UCBRFx	UCBRSx
ACLK (UCSSEL=01)	32.768 kHz	1200	1	1	11	0x25
	32.768 kHz	2400	13	0	-	0xB6
	32.768 kHz	4800	6	0	-	0xEE
	32.768 kHz	9600	3	0	-	0x92
SMCLK (UCSSEL=10 or UCSSEL=11)	1 MHz	9600	6	1	8	0x20
	1 MHz	19200	3	1	4	0x02
	1 MHz	38400	1	1	10	0x00
	1 MHz	57600	17	0	-	0x4A
	1 MHz	115200	8	0	-	0xD6

UCSSEL is configured in the **UCAxCTLW0** register.

This value is stored in the **UCAxBRW** register.

These bits are configured in the **UCAxMCTL** register.

EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

eUSCI_Ax Baud Rate Control Word (UCAxBRW) Register

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	UCAxBRW															
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
Reset:																

Bit	Field	Description
15:0	UCAxBRW	Clock Prescalar Settings of the Baud Rate Generator

Note: Only modify when UCSWRST=1.

UCA1BRW = 3;

eUSCI_Ax Modulation Control Word (UCAxMCTLW) Register

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	UCBRSx								UCBRFx				Reserved			UCOS16
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset:																

Bit	Field	Description
15:8	UCBRSx	Second Modulation Stage Select. These bits hold a free modulation pattern for BITCLK.
7:4	UCBRFx	First Modulation Stage Select. These bits determine the modulation pattern for BITCLK16 when UCOS16=1. Ignored when UCOS16=0. the "Oversampling Baud-Rate Generation" section of the MSP430 User's Guide shows the modulation pattern.
3:1	Reserved	Reserved
0	UCOS16	Oversampling Mode Enable (0=disabled; 1=enabled)

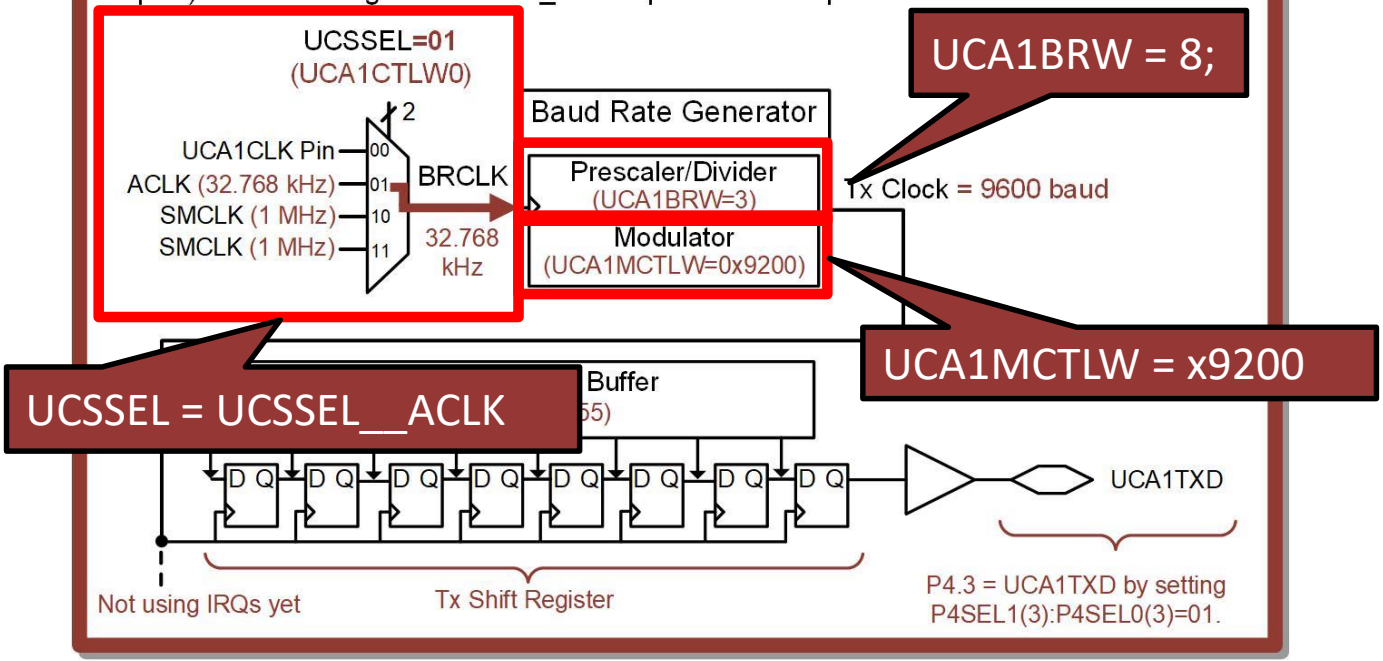
Note: Only modify when UCSWRST=1.

Notice:
- UCOS16 = 0 (low freq)
- UCBRFx = not used
- UCBRSx = x92

That's why
UCA1MCTLW = x9200

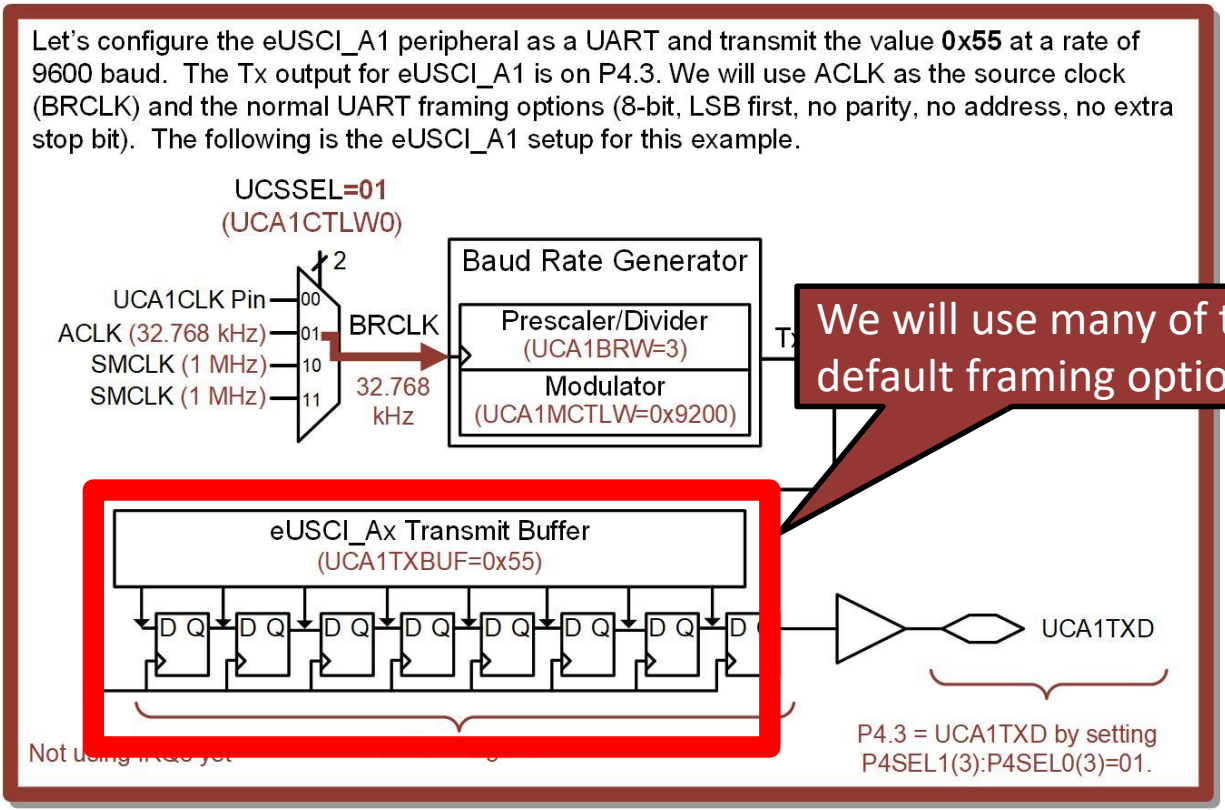
EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

Let's configure the eUSCI_A1 peripheral as a UART and transmit the value **0x55** at a rate of 9600 baud. The Tx output for eUSCI_A1 is on P4.3. We will use ACLK as the source clock (BRCLK) and the normal UART framing options (8-bit, LSB first, no parity, no address, no extra stop bit). The following is the eUSCI_A1 setup for this example.



EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

Let's configure the eUSCI_A1 peripheral as a UART and transmit the value **0x55** at a rate of 9600 baud. The Tx output for eUSCI_A1 is on P4.3. We will use ACLK as the source clock (BRCLK) and the normal UART framing options (8-bit, LSB first, no parity, no address, no extra stop bit). The following is the eUSCI_A1 setup for this example.



EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

The settings out of reset give us:

- No parity
- LSB First
- 8-Bit Length
- 1 Stop Bit

eUSCI_Ax Control Word Register 0 (UCAxCTLW0) – UART Mode

p:	15	14	13	12	11	10	9	8
	UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
Reset:	0	0	0	0	0	0	0	0

p:	7	6	5	4	3	2	1	0
	UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
Reset:	0	0	0	0	0	0	0	1

Bit	Field	Description
15	UCPEN	Parity Enable (0=Disabled; 1=Enabled)
14	UCPAR	Parity Select (0=Odd Parity; 1=Even Parity)
13	UCMSB	MSB First Select (0=LSB First; 1=MSB First)
12	UC7BIT	Character Length (0=8 bit; 1=7 bit)
11	UCSPB	Stop Bit Select (0=One Stop Bit; 1=Two Stop Bits)
10:9	UCMODEx	eUSCI_A Mode 00=UART Mode 10=Address-bit Multiprocessor Mode 01=Idle-Line Multiprocessor Mode 11=UART Mode w/ Auto Baud Detect
8	UCSYNC	Synchronous Mode Enable (0=Asynchronous Mode. UART; 1=Synchronous Mode. SPI)
7:6	UCSSELx	eUSCI_A Clock Source Select (BRCLK Source) 00=UCAxCLK Pin 10=SMCLK (1 MHz) 01=ACLK (32.768 kHz) 11=SMCLK (1 MHz)
5	UCRXEIE	Rx Erroneous-Character Interrupt Enable (0=Disabled; 1=Enabled)
4	UCBRKIE	Rx Break Character Interrupt Enable
3	UCDORM	Dormant. Put eUSCI_A into Sleep Mode (0=Not Dormat; 1=Dormat)
2	UCTXADDR	Transmit Address (0=Next Frame Transmitted is Data; 1=Next Frame Transmitted is Address.)
1	UCTXBRK	Tx Break (0=Next Frame Transmitted is not a break; 1=Next Frame Transmitted is a break or break/synch.)
0	UCSWRST	Software Reset Enable (0=Disabled. eUSCI operational; 1=Enabled. eUSCI held in reset)

EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

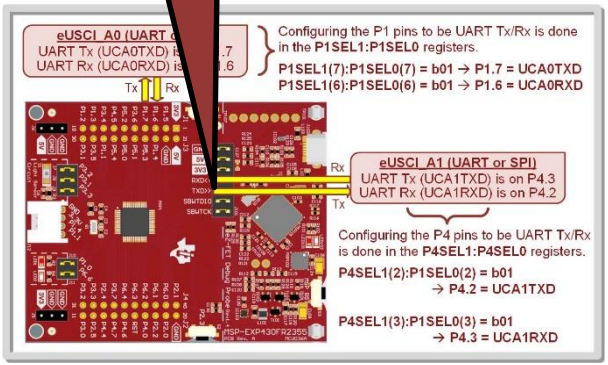
	PIN	UART	SPI	P1SEL1:0 Settings to Select eUSCI
eUSCI_A0	P1.7	Tx	SIMO	P1SEL1.7=0, P1SEL0.7 = 1
	P1.6	Rx	SOMI	P1SEL1.6=0, P1SEL0.6 = 1
	P1.5	-	SCLK	P1SEL1.5=0, P1SEL0.5 = 1
	P1.4	-	STE	P1SEL1.4=0, P1SEL0.4 = 1
eUSCI_A1	P4.3	Tx	SIMO	P4SEL1.3=0, P4SEL0.3 = 1
	P4.1	-	SCLK	P4SEL1.1=0, P4SEL0.1 = 1
	P4.0	-	STE	P4SEL1.0=0, P4SEL0.0 = 1
	PIN	I2C	SPI	P1SEL1:0 Settings to Select eUSCI
eUSCI_B0	P1.3	SCL	SIMO	P1SEL1.3=0, P1SEL0.3 = 1
	P1.2	SDA	SOMI	P1SEL1.2=0, P1SEL0.2 = 1
	P1.1	-	SCLK	P1SEL1.1=0, P1SEL0.1 = 1
	P1.0	-	STE	P1SEL1.0=0, P1SEL0.0 = 1
	PIN	I2C	SPI	P4SEL1:0 Settings to Select eUSCI
eUSCI_B1	P4.7	SCL	SIMO	P4SEL1.7=0, P4SEL0.7 = 1
	P4.6	SDA	SOMI	P4SEL1.6=0, P4SEL0.6 = 1
	P4.5	-	SCLK	P4SEL1.5=0, P4SEL0.5 = 1
	P4.4	-	STE	P4SEL1.4=0, P4SEL0.4 = 1

Configuring which eUSCI function to use within the peripheral is done using the UCSYNC bit in the **eUSCI_Ax Control Word Register 0** (UCAxCTLW0) or the UCMODEx bits in the **eUSCI_Bx Control Word Register 0** (UCBxCTLW0).

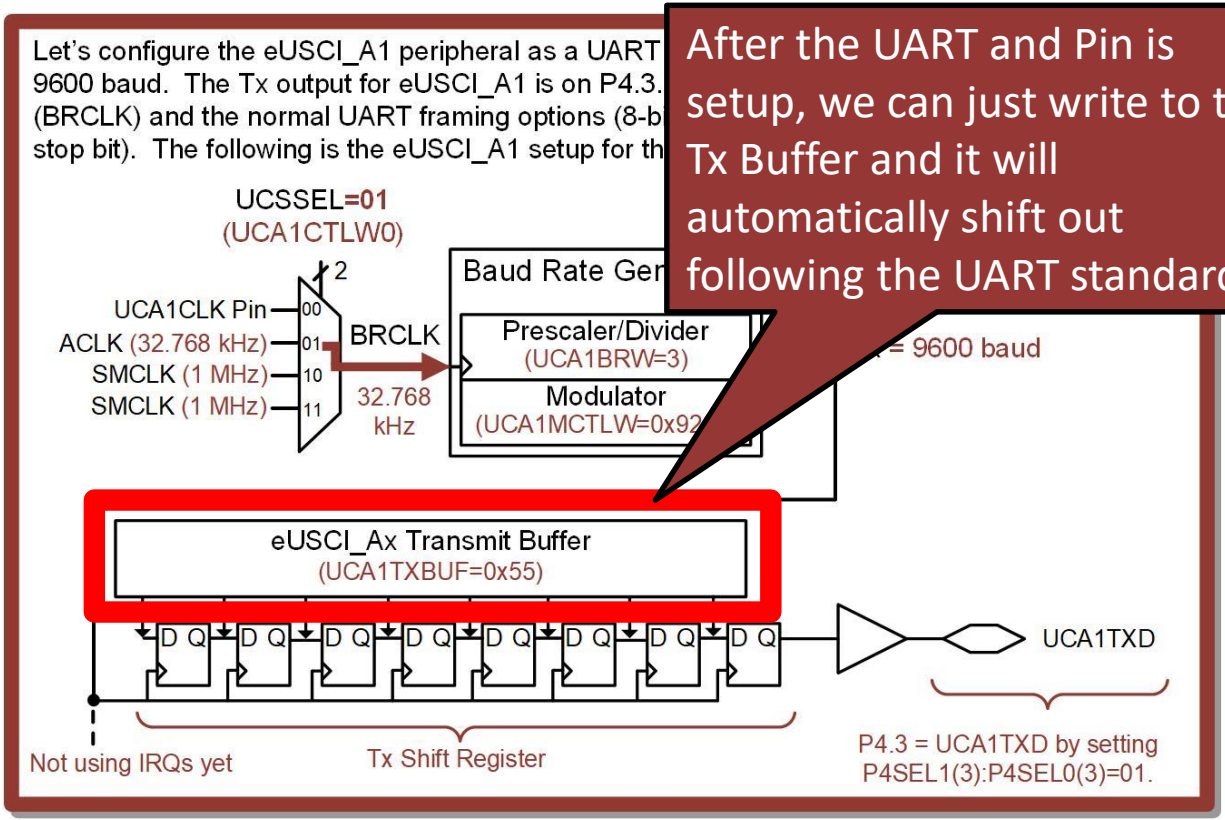
Configuring the pins to use the eUSCI capability is done using the **Port Function Select Registers** (PxSEL1:PPxSEL0).

The default setting for PxSEL1:PxSEL0=00, which selects the digital I/O port function for each pin.

A1 Tx is brought out to a pin on J101



EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD



EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

Step 1: In CCS, create a new C/C++ Empty Project (with main.c) titled:

C_UART_Tx2_Sending_Byte_at_9600

Step 2: Type in the following code in main.c after the statement to stop the watchdog timer.

Image Courtesy of https://neodem.wp.horizon.ac.uk/

EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

```c
#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;

    //-- 1. Put eUSCI_A1 into SW reset
    UCA1CTLW0 |= UCSWRST;

    //-- 2. Configure eUSCI_A1
    UCA1CTLW0 |= UCSSEL__ACLK;

    UCA1BRW = 3;
    UCA1MCTLW |= 0x9200;

    //-- 3. Configure Ports
    P4SEL1 &= ~BIT3;
    P4SEL0 |= BIT3;

    PM5CTL0 &= ~LOCKLPM5;

    //-- 4. Take eUSCI_A1 out of software reset
    UCA1CTLW0 &= ~UCSWRST;

  int i;

  while(1)
  {
      UCA1TXBUF = 0x55;
      for(i=0; i<10000; i=i+1){}
  }

  return 0;
}
```

Choose ACLK=32.768 kHz as BRCLK

N=32768/9600 = 3.41 → "Low Frequency Baud Rate Mode"
Prescaler → UCA1_BRW = 3
Modulation → UCA1MCTLW = 0x9200 (from table)

Put 0x55 in transmit buffer. As soon as it is stored, it will begin shifting out the data.

EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

Step 3: Debug your program and run it.

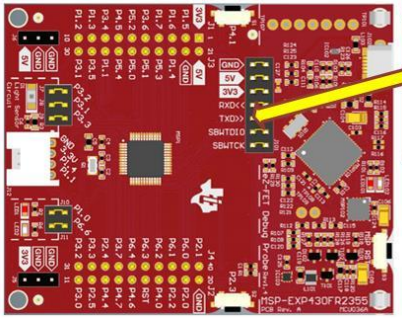
Note: We will first look at the Tx signal with an oscilloscope. If you don't have an oscilloscope, we will look at the signal in a different way in a later example.

EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD

Step 4: Observe the UART with an oscilloscope. Remove the jumper on the pins labeled “TXD>>” on the J101 header of the MSP430FR2355 board. Probe the pin nearest the MSP430 MCU (MSP1). This pin is being driven by UCA1TXD.

Step 5: Configure the oscilloscope waveform to center the UART frame. Measure the bit period of one of the short pulses.

EX: TRANSMITTING A BYTE OVER UART AT 9600 BAUD



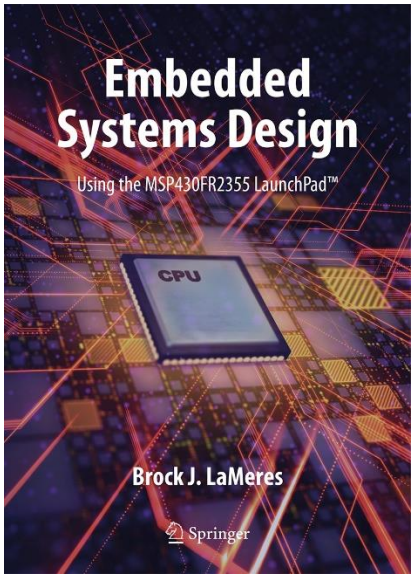
Remove this jumper and probe with an oscilloscope.

The bit period is ~121us, which corresponds to a ~9600 baud rate.

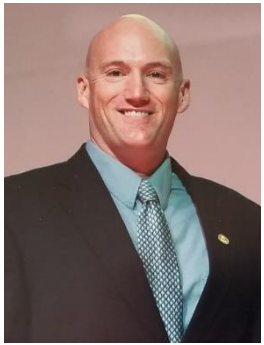
Did it work? Are you able to see the UART frame at 9600 baud? Notice that the sequence of bits received was 0b01010101. But remember that the UART sends the LSB first. Putting this in order from MSB we get: 0b10101010=0x55.

CHAPTER 14: SERIAL COMMUNICATIONS IN C

14.1.2 UART TRANSMIT ON THE MSP430FR2355 – EXAMPLE: TRANSMITTING A BYTE OVER UART AT 9600 BAUD



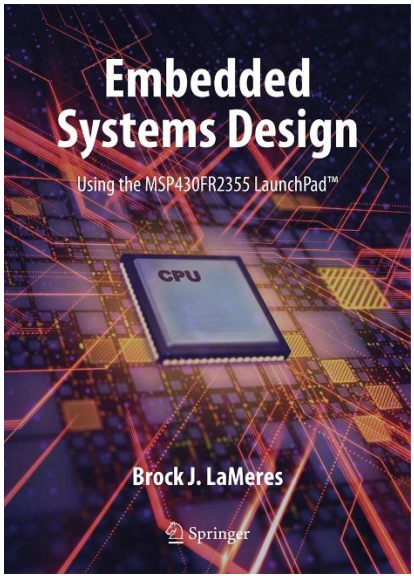
www.youtube.com/c/DigitalLogicProgramming_LaMeres

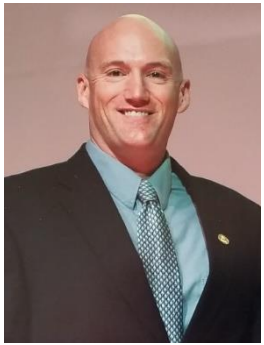


Embedded Systems Design

Chapter 14: Serial Communications in C

14.1.2 UART Transmit on the MSP430FR2355 – Example: Sending ASCII Characters to a Terminal

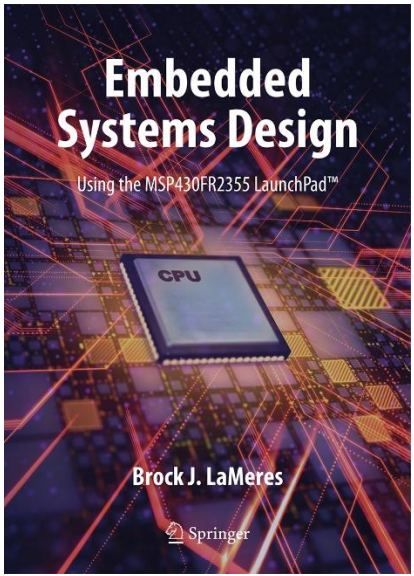


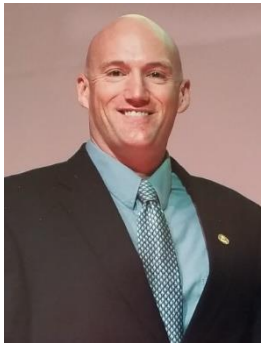


Embedded Systems Design

Chapter 14: Serial Communications in C

14.1.2 UART Transmit on the MSP430FR2355 – Example: Sending ASCII Characters to a Terminal





14.1.2 UART TRANSMIT ON THE MSP430FR2355

- **Terminal** – a way to communicate with a device using standard I/O on a computer.

- A terminal uses the UART interface to send and receive bytes through a serial port on a computer.

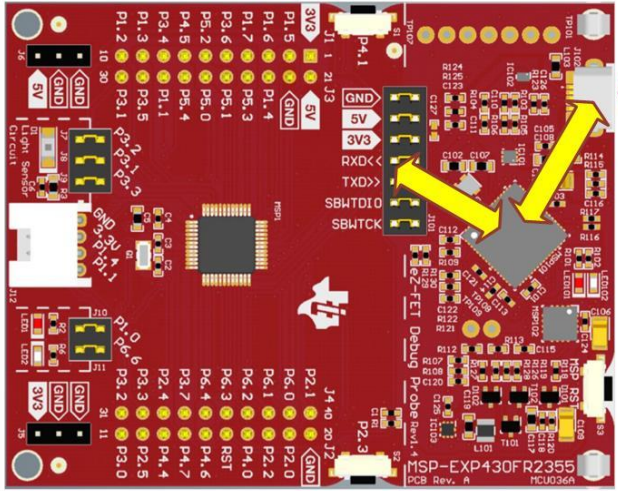
- The CCS design environment contains a built-in terminal feature that can be used to view data coming from the MSP430FR2355 board and also send information to the MCU board.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- The board contains a debug chip that handles downloading programs onto the MCU in addition to debugging programs.

- When setting up a terminal connection to the MSP430FR2355 board, the com port number of the Application UART1 is used as the connection ID

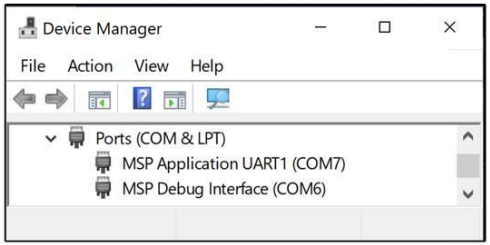
14.1.2 UART TRANSMIT ON THE MSP430FR2355



USB Com Port to Computer. The debugger IC on the LaunchPad allows the MSP430FR2355's eUSCI_A1 UART to appear as a com port to a computer. This is called the "MSP Application UART1" and will assigned a com port number by the computer.

On Windows, open "Device Manager" and view the "Ports (COM & LPT)". You will see two com ports. One is used by the debugger and one is the Application UART.

The Application UART allows a computer to interface with the MSP430FR2355 using a terminal window.



14.1.2 UART TRANSMIT ON THE MSP430FR2355

- **ASCII** – a coding scheme that uses terminal windows interface with the standard I/O of the computer
- **ASCII –** *American Standard Code for Information Interchange*
- This coding approach allows each button press on a computer's keyboard to result in an 8-bit code being generated and understood by the computer's CPU.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- When writing a program, the data type “char” can be used to declare variable and store characters in ASCII.

- In C, when a statement such as *char Var1 = ‘A’;* is used, it will assign the ASCII code for the symbol “A” into the 8-bit variable labeled “Var1.”

- Using the char datatype allows us to create programs that can send and receive ASCII characters using a terminal.

Image Courtesy of https://neodem.wp.horizon.ac.uk/

14.1.2 UART TRANSMIT ON THE MSP430FR2355

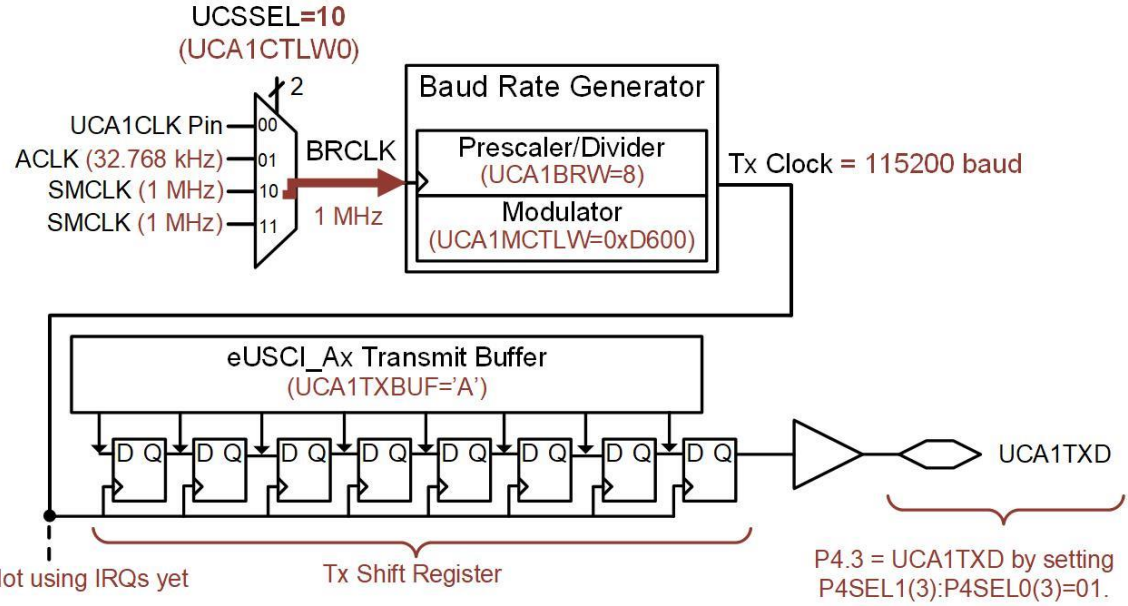
American Standard Code for Information Interchange (ASCII)

HEX	Char	Description	HEX	Char	Description
0x00	NUL	Null	0x40	@	At sign
0x01	SOH	Start of Header	0x41	A	Capital A
0x02	STX	Start of Text	0x42	B	Capital B
0x03	ETX	End of Text	0x43	C	Capital C
0x04	EOT	End of Transmission	0x44	D	Capital D
0x05	ENQ	Enquiry	0x45	E	Capital E
0x06	ACK	Acknowledge	0x46	F	Capital F
0x07	BEL	Bell	0x47	G	Capital G
0x08	BS	Backspace	0x48	H	Capital H
0x09	HT	Horizontal Tab	0x49	I	Capital I
0x0A	LF	Line Feed	0x4A	J	Capital J
0x0B	VT	Vertical Tab	0x4B	K	Capital K
0x0C	FF	Form Feed	0x4C	L	Capital L
0x0D	CR	Carriage Return	0x4D	M	Capital M
0x0E	SO	Shift Out	0x4E	N	Capital N
0x0F	SI	Shift In	0x4F	O	Capital O
0x10	DLE	Data Link Escape	0x50	P	Capital P
0x11	DC1	Device Control 1	0x51	Q	Capital Q
0x12	DC2	Device Control 2	0x52	R	Capital R
0x13	DC3	Device Control 3	0x53	S	Capital S
0x14	DC4	Device Control 4	0x54	T	Capital T
0x15	NAK	Negative Ack	0x55	U	Capital U
0x16	SYN	Synchronize	0x56	V	Capital V
0x17	ETB	End of Trans Block	0x57	W	Capital W
0x18	CAN	Cancel	0x58	X	Capital X
0x19	EM	End of Medium	0x59	Y	Capital Y
0x1A	SUB	Substitute	0x5A	Z	Capital Z
0x1B	ESC	Escape	0x5B	[	Left Square Bracket
0x1C	FS	File Separator	0x5C	\	Backslash
0x1D	GS	Group Separator	0x5D	]	Right Square Bracket
0x1E	RS	Record Separator	0x5E	^	Caret / Circumflex
0x1F	US	Unit Separator	0x5F	_	Underscore
0x20	space	Space	0x60	`	Grave / Accent
0x21	!	Exclamation Mark	0x61	a	Small a
0x22	"	Double Quote	0x62	b	Small b
0x23	#	Number	0x63	c	Small c
0x24	$	Dollar sign	0x64	d	Small d
0x25	%	Percent	0x65	e	Small e
0x26	&	Ampersand	0x66	f	Small f
0x27	'	Single Quote	0x67	g	Small g
0x28	(	Left Parenthesis	0x68	h	Small h
0x29	)	Right Parenthesis	0x69	i	Small i
0x2A	*	Asterisk	0x6A	j	Small j
0x2B	+	Plus	0x6B	k	Small k
0x2C	,	Comma	0x6C	l	Small l
0x2D	-	Minus	0x6D	m	Small m
0x2E	.	Period	0x6E	n	Small n
0x2F	/	Slash	0x6F	o	Small o
0x30	0	Zero	0x70	p	Small p
0x31	1	One	0x71	q	Small q
0x32	2	Two	0x72	r	Small r
0x33	3	Three	0x73	s	Small s
0x34	4	Four	0x74	t	Small t
0x35	5	Five	0x75	u	Small u
0x36	6	Six	0x76	v	Small v
0x37	7	Seven	0x77	w	Small w
0x38	8	Eight	0x78	x	Small x
0x39	9	Nine	0x79	y	Small y
0x3A	:	Colon	0x7A	z	Small z
0x3B	;	Semicolon	0x7B	{	Left Curly Bracket
0x3C	<	Less Than	0x7C	|	Vertical Bar
0x3D	=	Equality Sign	0x7D	}	Right Curly Bracket
0x3E	>	Greater Than	0x7E	~	Tilde
0x3F	?	Question Mark	0x7F	DEL	Delete

HEX	Char	Description
0x40	@	At sign
0x41	A	Capital A
0x42	B	Capital B
0x43	C	Capital C
0x44	D	Capital D
0x45	E	Capital E
0x46	F	Capital F
0x47	G	Capital G

EX: TRANSMITTING A CHARACTER OVER UART

Let's continually send the character 'A' from the eUSCI_A1 UART a rate of 115200 baud. We will first observe the character with an oscilloscope and then with a *terminal window* within CCS. The following is the eUSCI_A1 setup for this example.



EX: TRANSMITTING A CHARACTER OVER UART

Step 1: In CCS, create a new C/C++ Empty Project (with main.c) titled:

C_UART_Tx3_Sending_Char

Step 2: Type in the following code in main.c after the statement to stop the watchdog timer.

EX: TRANSMITTING A CHARACTER OVER UART

```
#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;

    //-- 1. Put eUSCI_A1 into SW reset
    UCA1CTLW0 |= UCSWRST;

    //-- 2. Configure eUSCI_A1
    UCA1CTLW0 |= UCSSEL__SMCLK;

    UCA1BRW = 8;
    UCA1MCTLW |= 0xD600;

    //-- 3. Configure Ports
    P4SEL1 &= ~BIT3;
    P4SEL0 |= BIT3;

    PM5CTL0 &= ~LOCKLPM5;

    //-- 4. Take eUSCI_A1 out of software reset
    UCA1CTLW0 &= ~UCSWRST;

  int i;

  while(1)
  {
      UCA1TXBUF = 'A';
      for(i=0; i<10000; i=i+1){}
  }

  return 0;
}
```

Put 'A' in transmit buffer using the type char. This will insert the ASCII code for the symbol 'A' (i.e., 0x41) into the buffer.

EX: TRANSMITTING A CHARACTER OVER UART

Step 3: Debug your program and run it.

Note: We will first look at the Tx signal with an oscilloscope to view the ASCII code for ‘A’. We will then look at the character using a terminal window.

Image Courtesy of https://neodem.wp.horizon.ac.uk/

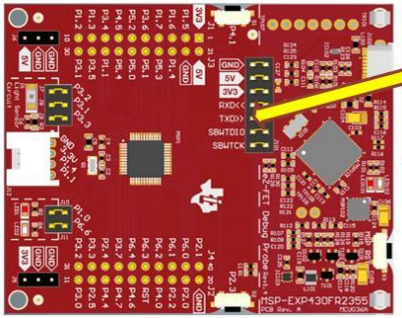
EX: TRANSMITTING A CHARACTER OVER UART

Step 4: Observe the UART with an oscilloscope. Remove the jumper on the pins labeled “TXD>>” on the J101 header of the MSP430FR2355 board. Probe the pin nearest the MSP430 MCU (MSP1). This pin is being driven by UCA1TXD.

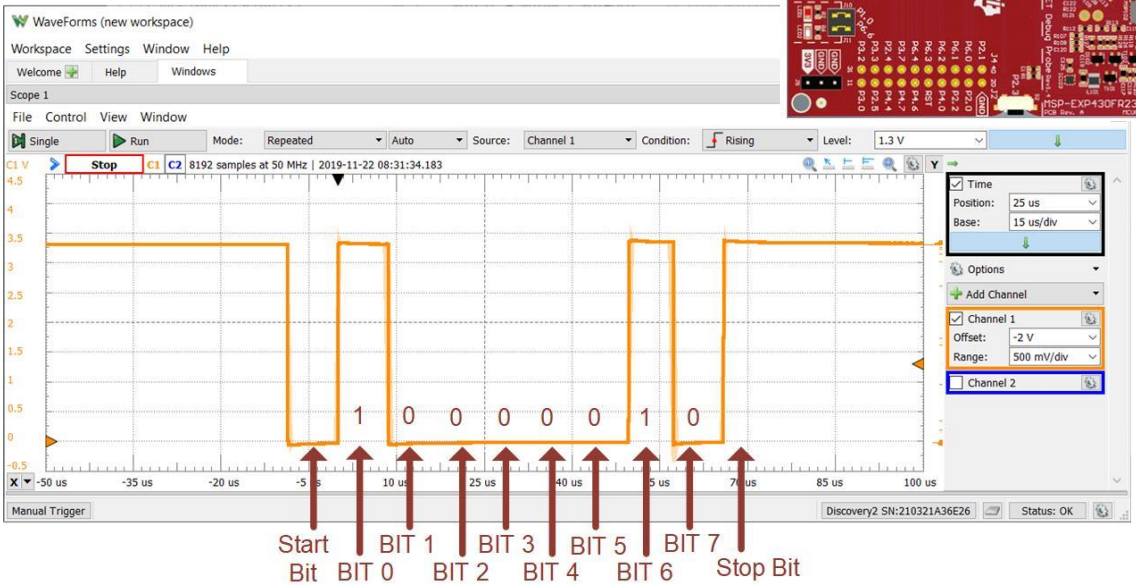
Step 5: Configure the oscilloscope waveform to center the UART frame.

Image Courtesy of https://neodem.wp.horizon.ac.uk/

EX: TRANSMITTING A CHARACTER OVER UART



Remove this jumper and probe with an oscilloscope.



Did it work? You should have seen a bit stream arrive of 10000010. After rearranging this to have the MSB first, you get 01000001=0x41. This is the ASCII character for 'A' that was put into the Tx Buffer.

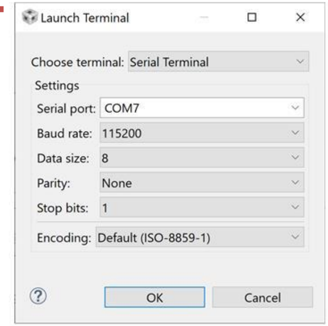
EX: TRANSMITTING A CHARACTER OVER UART

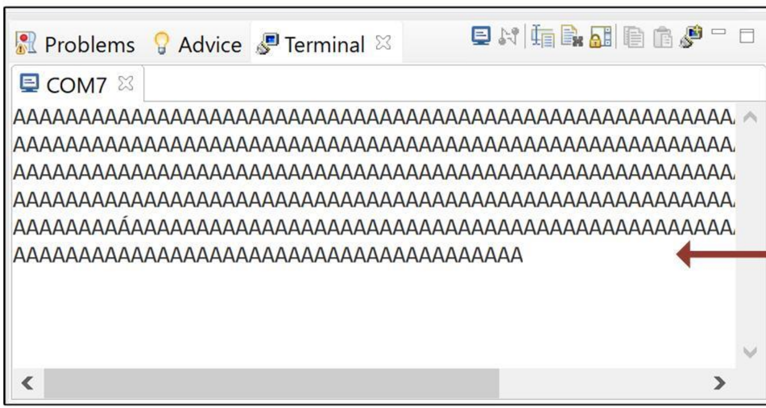
Step 6: Now let's launch a terminal within CCS and observe the ASCII character arrive. In CCS, use the pull down menu View → Terminal. A terminal tab will appear within CCS (usually in the lower right corner along with the "Problems" and "Advice" tabs).

Step 7: Configure the terminal window. Click on the "Open a Terminal" button on the terminal tab. This will bring up a configuration window. You will need to determine the Application UART1's com port on your computer. On Windows, this can be found using "Device Manager → Ports (COM & LPT)". Configure the terminal to use this com port number with a baud rate of 115200, a data size of 8, no parity, and 1 stop bit. Leave the encoding at its default value.

EX: TRANSMITTING A CHARACTER OVER UART

Step 8: Plug your jumper back into the MSP430FR2355 board and observe the character ‘A’ arrive.





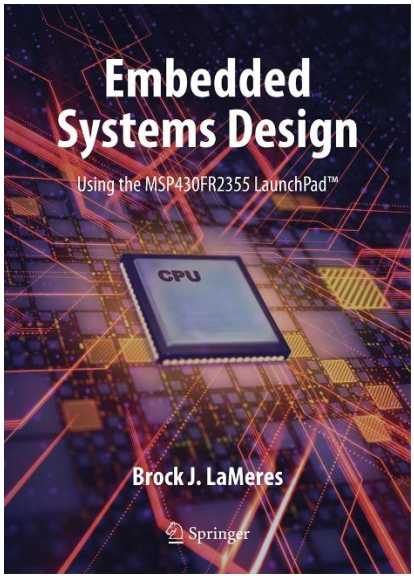
Note that since the Application UART1 shares the USB cable with the debugger, the terminal window can be sluggish when in debug mode. You can “Terminate” the debugging session in CCS and your program will still be running on the MCU. Do this to observe the real-time operation of the UART data arriving.

Did it work? You should see the character ‘A’ continually be printed in the terminal window.

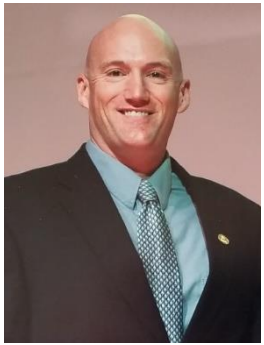
Embedded Systems Design

Chapter 14: Serial Communications in C

14.1.2 UART Transmit on the MSP430FR2355 – Example: Sending ASCII Characters to a Terminal

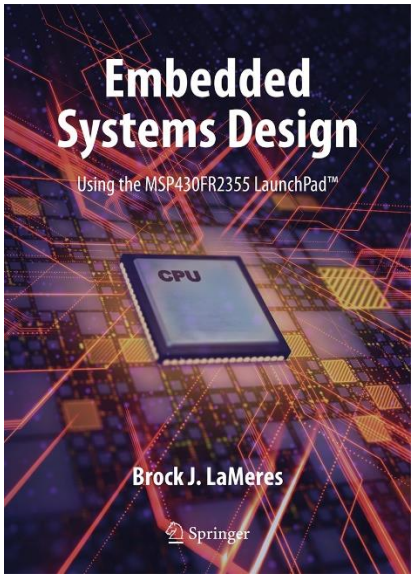


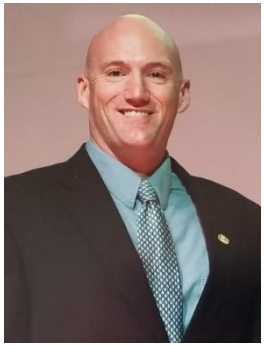
www.youtube.com/c/DigitalLogicProgramming_LaMeres



CHAPTER 14: SERIAL COMMUNICATIONS IN C

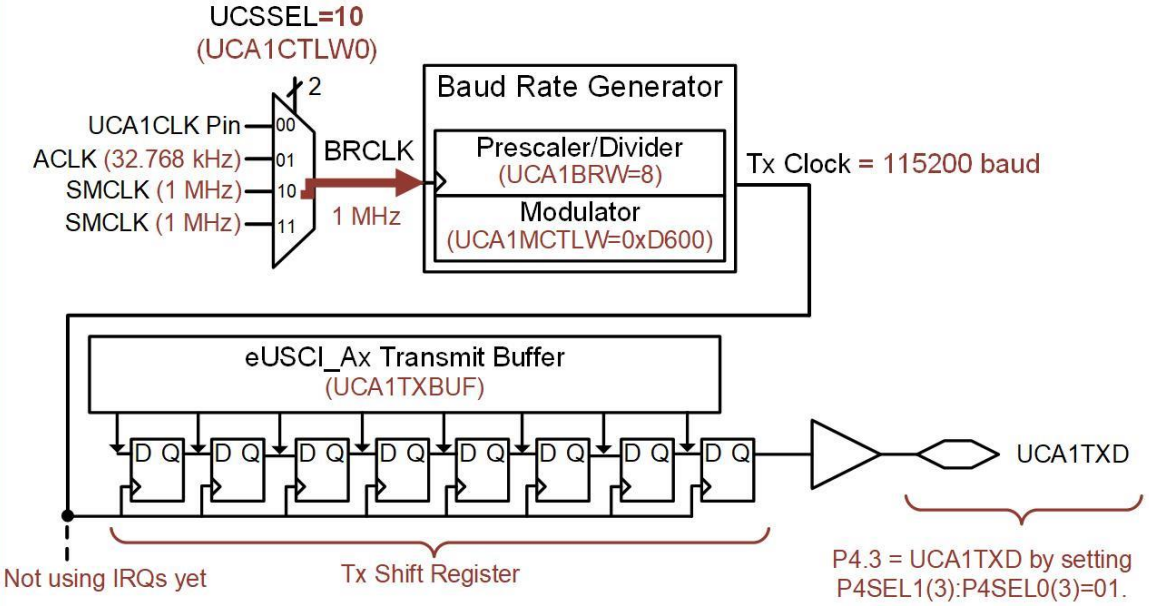
14.1.2 UART TRANSMIT ON THE MSP430FR2355 – EXAMPLE: SENDING A STRING TO A TERMINAL





EX: TRANSMITTING A STRING OVER UART

Let's continually send the string "Hello World " from the eUSCI_A1 UART a rate of 115200 baud. We will observe the string with the *terminal window* within CCS. The following is the eUSCI_A1 setup for this example.



EX: TRANSMITTING A STRING OVER UART

Step 1: In CCS, create a new C/C++ Empty Project (with main.c) titled:

C_UART_Tx4_Sending_String

Step 2: Type in the following code in main.c after the statement to stop the watchdog timer.

EX: TRANSMITTING A STRING OVER UART

```
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;

    //-- 1. Put eUSCI_A1 into SW reset
    UCA1CTLW0 |= UCSWRST;

    //-- 2. Configure eUSCI_A1
    UCA1CTLW0 |= UCSSEL__SMCLK;

    UCA1BRW = 8;
    UCA1MCTLW |= 0xD600;

    //-- 3. Configure Ports
    P4SEL1 &= ~BIT3;
    P4SEL0 |= BIT3;

    PM5CTL0 &= ~LOCKLPM5;

    //-- 4. Take eUSCI_A1 out of SW reset
    UCA1CTLW0 &= ~UCSWRST;

  char message[] = "Hello World ";
  int position;
  int i, j;

  while(1)
  {
      for (position=0; position<sizeof(message); position++)
      {
         UCA1TXBUF = message[position];
         for(i=0; i<100; i=i+1){} // delay between chars
      }

      for(j=0; j<30000; j=j+1){}  // delay between strings
  }

  return 0;
}
```

The variable "message" will be a string of ASCII characters initialized with "Hello World ".

The variable position will be used to access the individual characters within the string.

The for() loop will send each character in the string "message" one-by-one.

We'll add a short delay between character transmissions.

We'll add a longer delay between string transmissions.

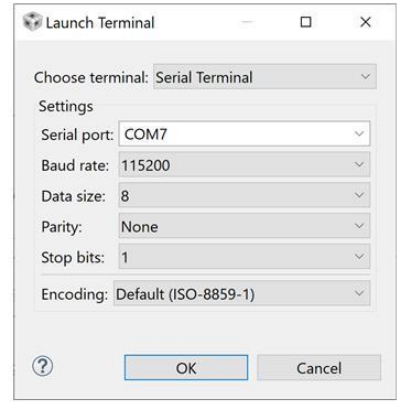
EX: TRANSMITTING A STRING OVER UART

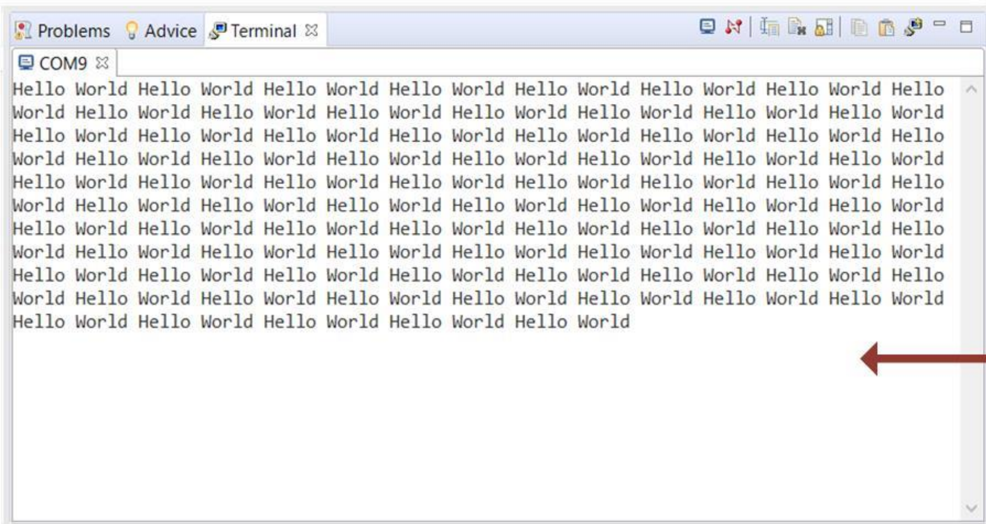
Step 3: Debug your program and run it.

Note: We will first look at the Tx string using the terminal window within CCS.

Step 4: Launch a terminal within CCS and observe the string arrive continually. Configure the terminal to use the com port of the Application UART1 with a baud rate of 115200, a data size of 8, no parity, and 1 stop bit. Leave the encoding at its default value. You should see “Hello World” arrive continually on the terminal.

Ex: Transmitting a String Over UART





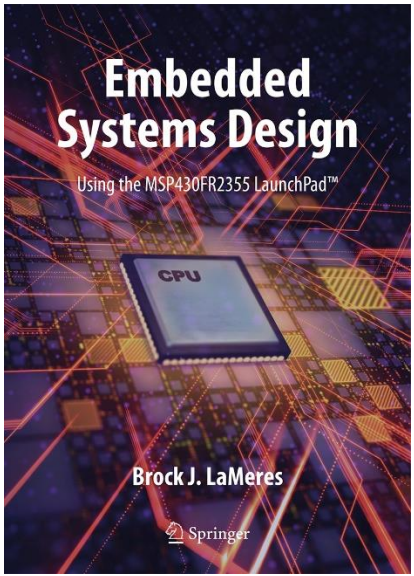
If it is running sluggish, terminate the debug session so that only the Application UART1 is communicating with your computer over the USB connection.

?

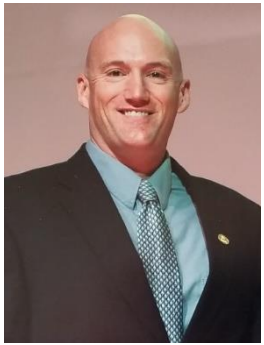
Did it work? You should see “Hello World “ be continually printed to the terminal window.

CHAPTER 14: SERIAL COMMUNICATIONS IN C

14.1.2 UART TRANSMIT ON THE MSP430FR2355 – EXAMPLE: SENDING A STRING TO A TERMINAL



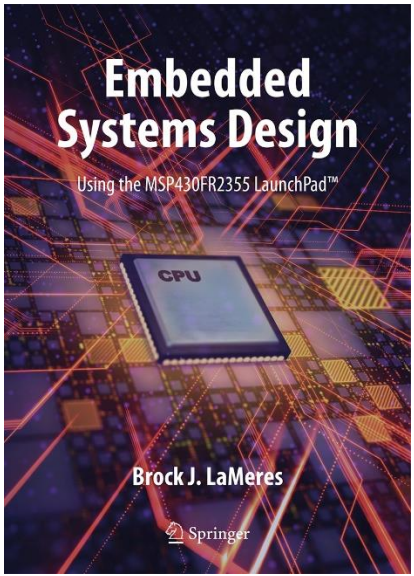
www.youtube.com/c/DigitalLogicProgramming_LaMeres

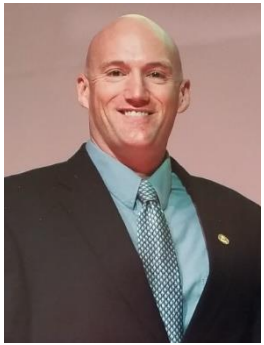


Embedded Systems Design

Chapter 14: Serial Communications in C

14.1.2 UART Transmit on the MSP430FR2355 – Example: Using Tx IRQs to Control UART Transmission





14.1.2 UART TRANSMIT ON THE MSP430FR2355

- One of the drawbacks of the prior examples was that delay loops were needed to space out when data was put into the Tx buffer.
- The eUSCI UART contains a variety of flags that aid in monitoring the status of the link.
- These flags can also trigger IRQs.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

eUSCI_Ax Interrupt Flag (UCAxIFG) Register – UART Mode

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved												UCTXCPTIEG	UCSTTIEG	UCTXIFG	UCRXIFG
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Bit	Field	Description
15:4	Reserved	-
3	UCTXCPTIFG	Transmit Complete Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending) UCTXCPTIFG is set when the entire byte in the internal shift register is shifted out and UCAxTXBUF is empty.
2	UCSTTIFG	Start Bit Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending) UCSTTIFG is set after a Start bit was received.
1	UCTXIFG	Transmit Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending) UCTXIFG is set when UCAxTXBUF is empty.
0	UCRXIFG	Receive Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending) UCRXIF is set when UCAxRXBUF has received a complete character.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- **Transmit Interrupt (UCTXIFG):**
  - Flag is cleared when the Tx buffer data is written to.
  - Flag is set when the Tx buffer data has been moved to the shift register and the buffer is empty.
  - Only indicates that the Tx buffer is empty, not that the last character has been fully shifted out.

Image Courtesy of https://neodem.wp.horizon.ac.uk/

14.1.2 UART TRANSMIT ON THE MSP430FR2355

- **Transmit Complete Interrupt (UCTXCPTIFG):**
  - Flag is set when the entire byte in in Tx shift register has been shifted out and it is ready for a new character.

- The two Tx IRQs are enabled with the UCTXIE and UCTXCPTIE bits with in the UCAxIE

- The eUSCI_Ax peripheral only contains one interrupt vector address for all the interrupts within the system.

14.1.2 UART TRANSMIT ON THE MSP430FR2355

eUSCI_Ax Interrupt Enable (UCAxIE) Register – UART Mode

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved												UCTXCPTIE	UCSTTIE	UCTXIE	UCRXIE
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:4	Reserved	-
3	UCTXCPTIE	Transmit Complete Interrupt Enable (0=Disabled; 1=Enabled)
2	UCSTTIE	Start Bit Interrupt Enable (0=Disabled; 1=Enabled)
1	UCTXIE	Transmit Interrupt Enable (0=Disabled; 1=Enabled)
0	UCRXIE	Receive Interrupt Enable (0=Disabled; 1=Enabled)

14.1.2 UART TRANSMIT ON THE MSP430FR2355

eUSCI_Ax Interrupt Vector (UCAxIV) Register – UART Mode

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	UCIVx															
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:0	UCIVx	eUSCI_A Interrupt Vector Value 00h = No IRQ Pending 02h = Interrupt Source → Rx Buffer Full; Interrupt Flag → UCRXIFG (highest priority) 04h = Interrupt Source → Tx Buffer Empty; Interrupt Flag → UCTXIFG 06h = Interrupt Source → Start Bit Received; Interrupt Flag → UCSTTIFG 08h = Interrupt Source → Transmit Complete; Interrupt Flag → UCTXCPTIFG (lowest priority)

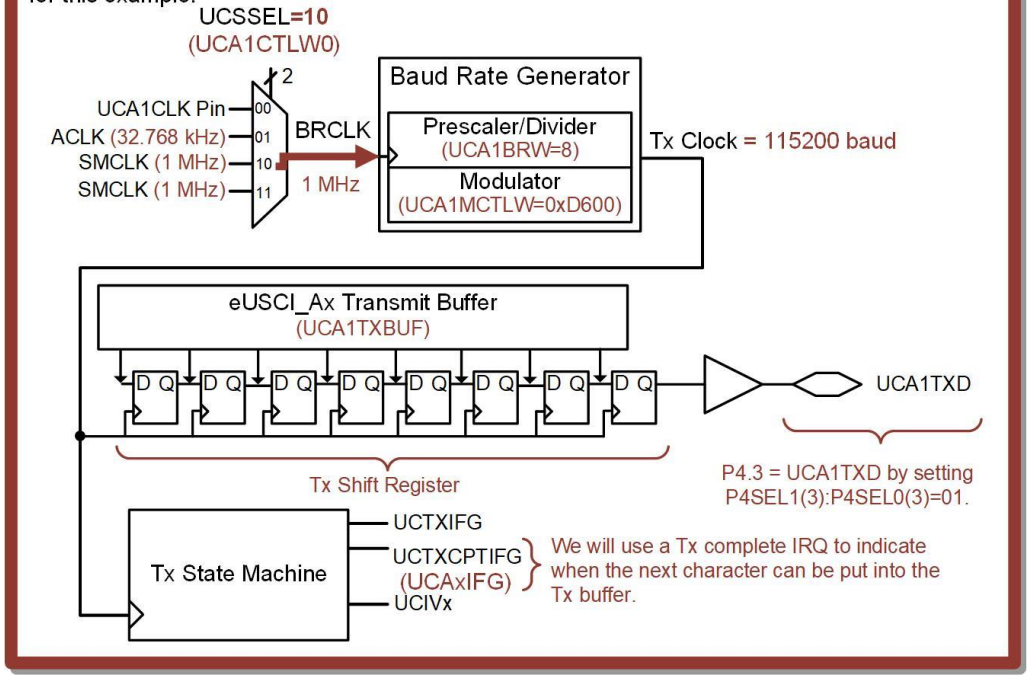
14.1.2 UART TRANSMIT ON THE MSP430FR2355

INTERRUPT SOURCE	INTERRUPT FLAG	INTERRUPT TYPE	VECTOR ADDR	VECTOR LABEL
System Reset	SVSHIFG, PMMRSTIFG, WDTIFG, PMMPORIFG, PMMBORIFG, SYSRSTIV, FLLULPUC	Reset	FFFEh	RESET_VECTOR
System NMI	VMAIFG, JMBINIFG, JMBOUTIFG, CBDIFG, UBDIFG	Non-Maskable	FFFCh	SYSNMI_VECTOR
User NMI	NMIIFG, OFIFG	Non-Maskable	FFFAh	UNMI_VECTOR
Timer0_B3	TB0CCR0 CCIFG0	Maskable	FFF8h	TIMER0_B0_VECTOR
Timer0_B3	TB0CCR1 CCIFG1, TB0CCR2 CCIFG2, TB0IFG (TB0IV)	Maskable	FFF6h	TIMER0_B1_VECTOR
Timer1_B3	TB1CCR0 CCIFG0	Maskable	FFF4h	TIMER1_B0_VECTOR
Timer1_B3	TB1CCR1 CCIFG1, TB1CCR2 CCIF TB1IFG (TB1IV)			
Timer2_B3	TB2CCR0 CCIFG0			
Timer2_B3	TB2CCR1 CCIFG1, TB2CCR2 CCIF TB2IFG (TB2IV)			
Timer3_B7	TB3CCR0 CCIFG0	M	Ch	TIMER3_B0_VECTOR
Timer3_B7	TB3CCR1 CCIFG1, TB3CCR2 CCIFG2, TB3CCR3 CCIFG3, TB3CCR4 CCIFG4, TB3CCR5 CCIFG5, TB3CCR6 CCIFG6, TB3IFG (TB3IV)		FFEAh	TIMER3_B1_VECTOR
RTC counter	RTCIFG	Maskable	FFE8h	RTC_VECTOR
eUSCI_A0 (Rx or Tx)	UCTXCPTIFG, UCSTTIFG, UCRXIFG, UCTXIFG (UART mode) UCRXIFG, UCTXIFG (SPI mode) (UCA0IV))	Maskable	FFE4h	EUSCI_A0_VECTOR
eUSCI_A1 (Rx or Tx)	UCTXCPTIFG, UCSTTIFG, UCRXIFG, UCTXIFG (UART mode) UCRXIFG, UCTXIFG (SPI mode) (UCA0IV)	Maskable	FFE2h	EUSCI_A1_VECTOR

Each of the 4x eUSCI systems have a dedicate interrupt vector.

EX: CONTROLLING UART TRANSMISSION WITH INTERRUPTS

Let's design a program that will send the string "Hello World " from the eUSCI_A1 UART each time S1 is pressed on the LaunchPad™ board. We will observe the string with a terminal window within CCS. We will use a Tx *complete interrupt* (UCTXCPTIFG) to control when the next character of the string is inserted into the Tx buffer. The following is the eUSCI_A1 setup for this example.



EX: CONTROLLING UART TRANSMISSION WITH INTERRUPTS

Step 1: In CCS, create a new C/C++ Empty Project (with main.c) titled:

C_UART_Tx5_Sending_String_using_UCTXCPTIFG

Step 2: Type in the following code in main.c after the statement to stop the watchdog timer.

EX: CONTROLLING UART TRANSMISSION WITH INTERRUPTS

```
#include <msp430.h>

char message[] = "Hello World "
unsigned int position;

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;
    //-- 1. Put eUSCI_A1 into software reset
    UCA1CTLW0 |= UCSWRST;
    //-- 2. Configure eUSCI_A1
    UCA1CTLW0 |= UCSSEL__SMCLK;
    UCA1BRW = 8;
    UCA1MCTLW |= 0xD600;
    //-- 3. Configure Ports
    P4DIR &= ~BIT1;
    P4REN |= BIT1;
    P4OUT |= BIT1;
    P4IES |= BIT1;
    P4SEL1 &= ~BIT3;
    P4SEL0 |= BIT3;
    PM5CTL0 &= ~LOCKLPM5;
    //-- 4. Take eUSCI_A1 out of SW reset
    UCA1CTLW0 &= ~UCSWRST;
    //-- 5. Enable IRQs
    P4IFG &= ~BIT1;
    P4IE |= BIT1;
    __enable_interrupt();

    while(1){}
    return 0;
}
```

Since we will be accessing the string and the index variables from multiple service routines, they will be defined as global variables before the int main(void) routine.

Since the variable "position" will be used as an index to the string held in "message", we will define it as type "unsigned int" so that it never takes on a negative value.

We need to setup P4.1 to handle the button press on S1.

In our initialization we only enable the S1 port IRQ. Enabling the eUSCI_A1 IRQ and clearing the UCTXCPTIE flag will take place in the interrupt service routines.

The Interrupt Service Routines are shown in the next example figure.

EX: CONTROLLING UART TRANSMISSION WITH INTERRUPTS

```
//-- Interrupt Service Routines ------------
#pragma vector = PORT4_VECTOR
__interrupt void ISR_Port4_S1(void)
{
    position = 0;
    UCA1IE |= UCTXCPTIE;
    UCA1IFG &= ~UCTXCPTIFG;
    UCA1TXBUF = message[position];

    P4IFG &= ~BIT1;
}
```

When S1 is pressed, the ISR will initialize the string index variable "position" to the location of the first character in the string.

Next, it enables the UCTXCPTIE IRQ and clears the UCTXCPTIFG flag.

It then puts the first character of the string into the Tx buffer.

Once the first character is placed in the Tx buffer, the UART will start shifting it out. Once the shifting is complete and the buffer is ready for the next character, the UCTXCPTIFG IRQ will trigger.

```
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void)
{
    if(position == sizeof(message)) {
       UCA1IE &= ~UCTXCPTIE;
    }
    else {
       position++;
       UCA1TXBUF = message[position];
    }
    UCA1IFG &= ~UCTXCPTIFG;
}
```

When this routine triggers for the first time, the first character has already been sent.

First, check whether all of the characters in the string have been sent. If they have, disable the UCTXCPTIE IRQ.

If more characters need to be sent, increment the position index and place the next character in the Tx buffer.

At the end of the ISR we need to clear the UCTXCPTIFG flag.

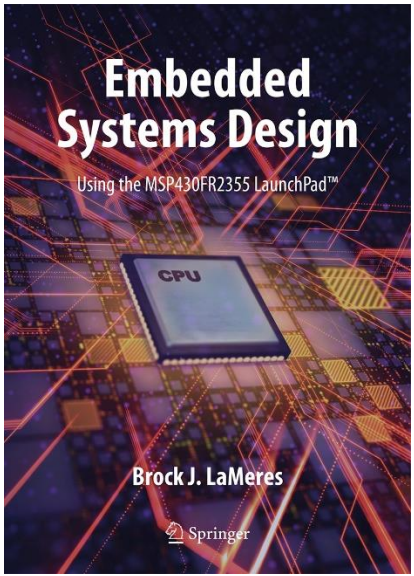
EX: CONTROLLING UART TRANSMISSION WITH INTERRUPTS

Step 3: Save and debug your program. Run and then terminate your program. Your program will be running but CCS will not be sharing the USB connection between the debugger and Application UART1.

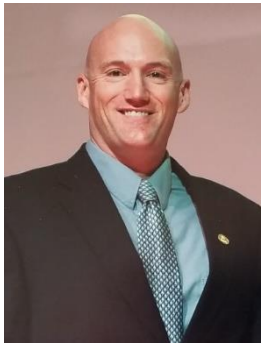
Step 4: Launch a terminal within CCS and configure it to communicate with the MSP430FR2355 board.

CHAPTER 14: SERIAL COMMUNICATIONS IN C

14.1.2 UART TRANSMIT ON THE MSP430FR2355 – EXAMPLE: USING TX IRQS TO CONTROL UART TRANSMISSION

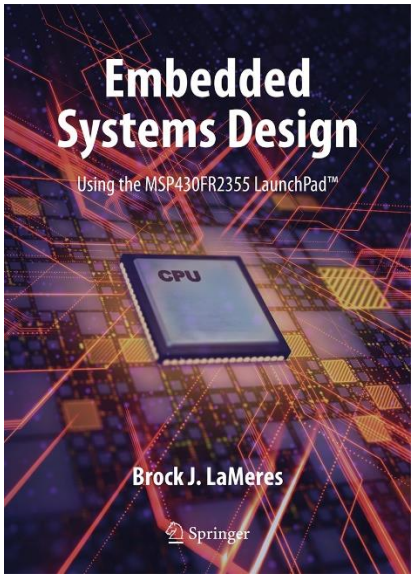


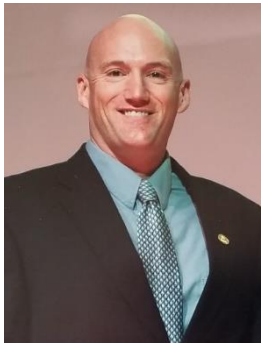
www.youtube.com/c/DigitalLogicProgramming_LaMeres



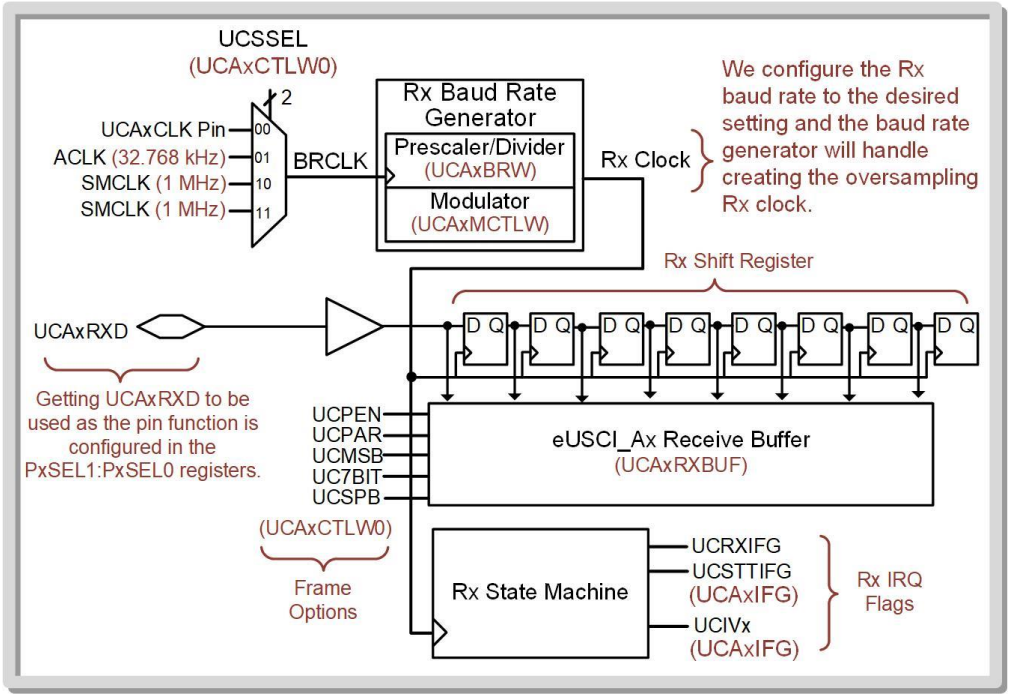
CHAPTER 14: SERIAL COMMUNICATIONS IN C

14.1.3 UART RECEIVE ON THE MSP430FR2355 – OVERVIEW



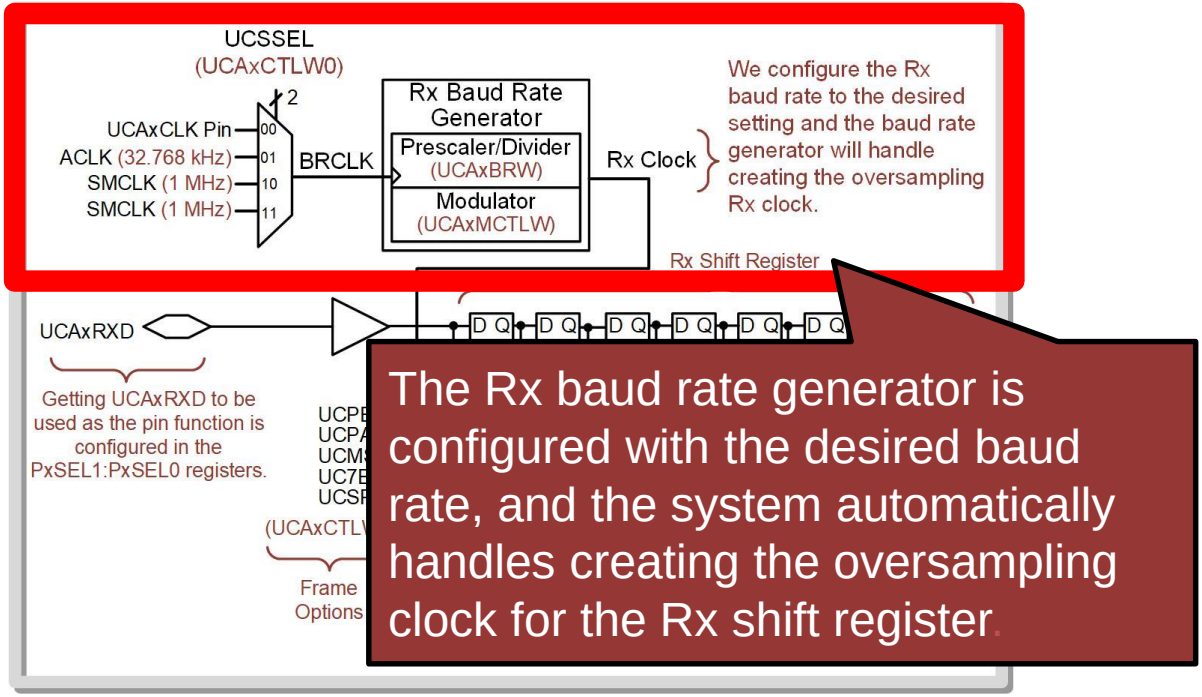


14.1.3 UART RECEIVE ON THE MSP430FR2355



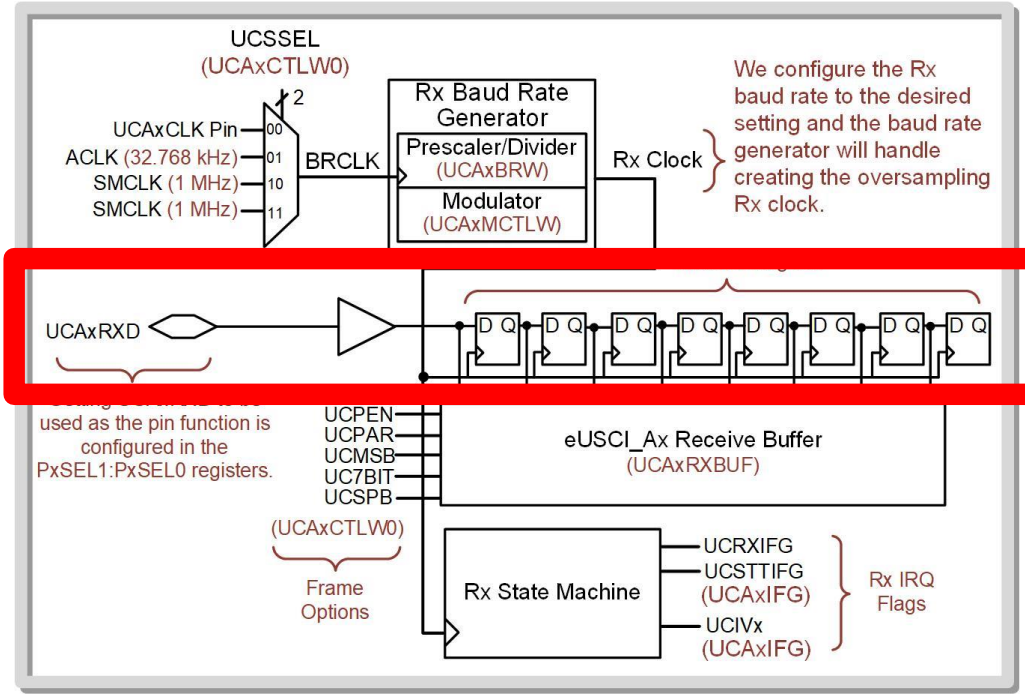
14.1.3 UART RECEIVE ON THE MSP430FR2355

- The same clock generator is used for both the Tx and Rx circuits and is configured using the UCAxBRW and UCAxMCTLW registers.



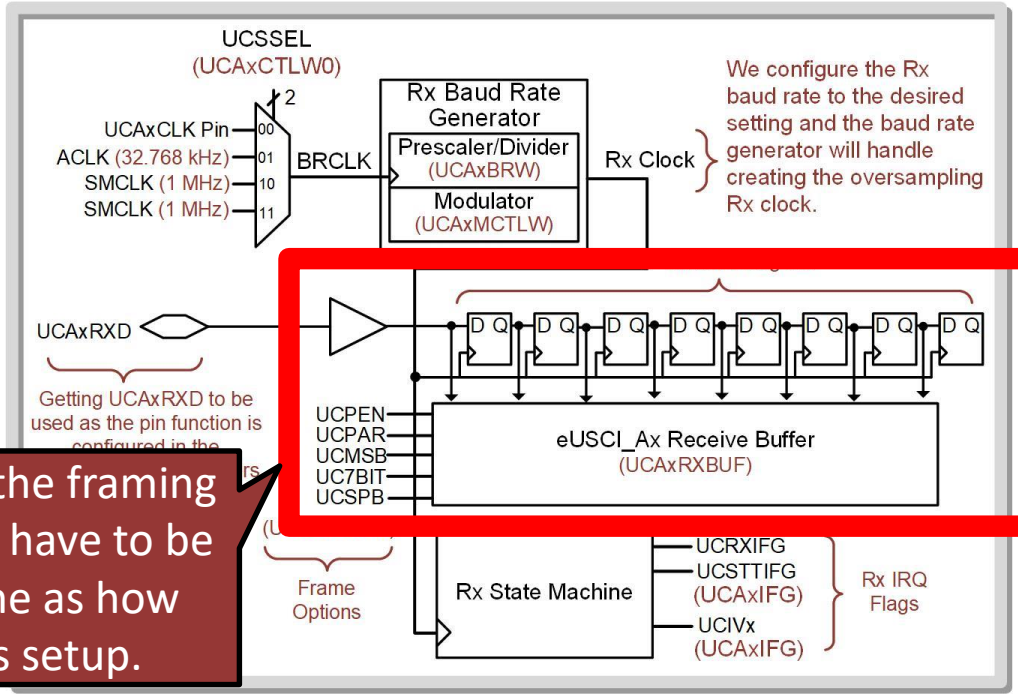
14.1.3 UART RECEIVE ON THE MSP430FR2355

- The Rx system is very similar to the Tx system in that it contains a shift register that receives the serial data.



14.1.3 UART RECEIVE ON THE MSP430FR2355

- Once the frame has been received, it is put into the Rx receiver buffer (UCAxRXBUF).



Notice the framing options have to be the same as how the Tx is setup.

14.1.3 UART RECEIVE ON THE MSP430FR2355

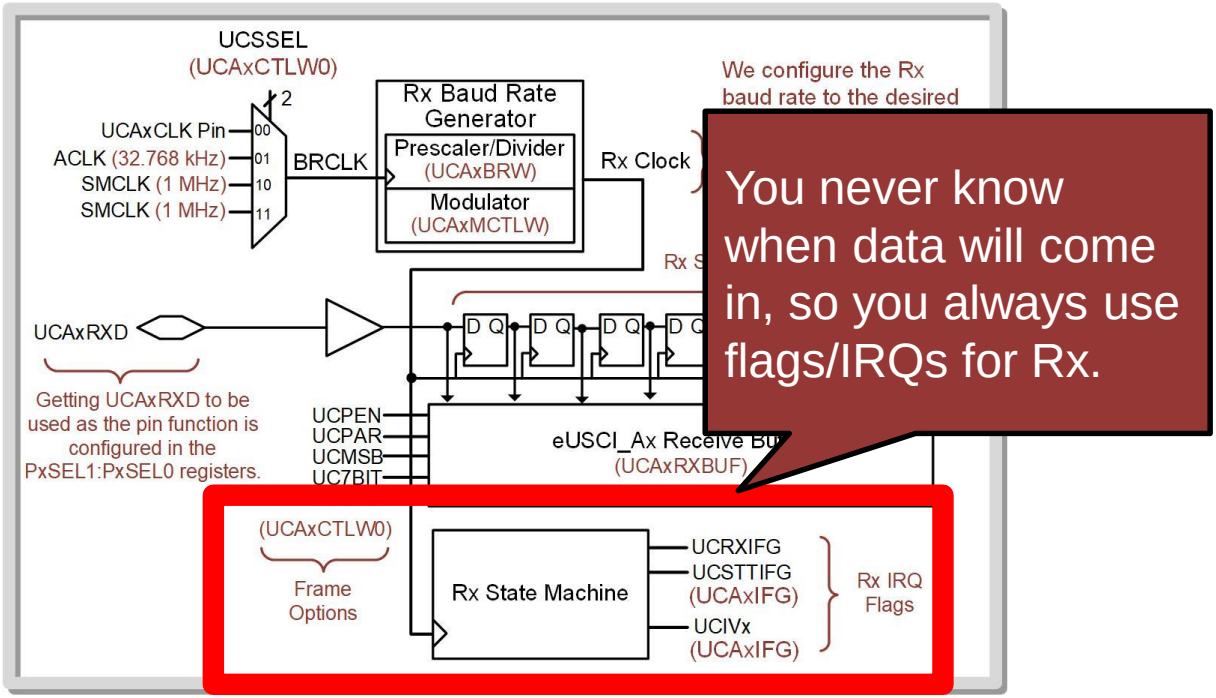
eUSCI_Ax Receive Buffer (UCAxRXBUF) Register

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								UCRXBUFx							
Value on Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:8	Reserved	-
7:0	UCRXBUFx	This register holds the last data received by the Rx shift register. Reading this register resets the receive-error bits, the UCADDR/UCIDLE bit, and the UCRXIFG flag.

14.1.3 UART RECEIVE ON THE MSP430FR2355

- The Rx system has a state machine that monitors the incoming data and creates status flags that can be used to generate interrupts.



14.1.3 UART RECEIVE ON THE MSP430FR2355

- **Receive interrupt (UCRXIFG):**
  - Triggers when a new character has been received and the data is available in the Rx buffer.
  - Cleared when the Rx buffer is read.
  - UCRXIFG has its own flag and local IRQ enable.

eUSCI_Ax Interrupt Flag (UCAxIFG) Register – UART Mode

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved												UCTXCPTIEG	UCSTTIEG	UCTXIFG	UCRXIFG
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Bit	Field	Description
15:4	Reserved	-
3	UCTXCPTIFG	Transmit Complete Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending) UCTXCPTIFG is set when the entire byte in the internal shift register is shifted out and UCAxTXBUF is empty.
2	UCSTTIFG	Start Bit Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending) UCSTTIFG is set after a Start bit was received.
		Transmit Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending)
0	UCRXIFG	Receive Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending) UCRXIF is set when UCAxRXBUF has received a complete character.

eUSCI_Ax Interrupt Enable (UCAxIE) Register – UART Mode

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved												UCTXCPTIE	UCSTTIE	UCTXIE	UCRXIE
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:4	Reserved	-
3	UCTXCPTIE	Transmit Complete Interrupt Enable (0=Disabled; 1=Enabled)
2	UCSTTIE	Start Bit Interrupt Enable (0=Disabled; 1=Enabled)
0	UCRXIE	Receive Interrupt Enable (0=Disabled; 1=Enabled)

14.1.3 UART RECEIVE ON THE MSP430FR2355

- **Start bit interrupt (UCSTTIFG)**
  - Triggers when the system sees a HIGH-to-LOW transition on the Rx pin indicating a new frame is being received.
  - UCSTTIFG has its own flag and local IRQ enable.

eUSCI_Ax Interrupt Flag (UCAxIFG) Register – UART Mode

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved												UCTXCPTIEG	UCSTTIEG	UCTXIFG	UCRXIFG
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Bit	Field	Description
15:4	Reserved	-
		Transmit Complete Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending)
2	UCSTTIFG	Start Bit Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending) UCSTTIFG is set after a Start bit was received.
		UCTXIFG is set when UCAxTXBUF is empty.
0	UCRXIFG	Receive Interrupt Flag (0=No IRQ Pending; 1=IRQ Pending) UCRXIF is set when UCAxRXBUF has received a complete character.

eUSCI_Ax Interrupt Enable (UCAxIE) Register – UART Mode

p:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved												UCTXCPTIE	UCSTTIE	UCTXIE	UCRXIE
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Field	Description
15:4	Reserved	-
2	UCSTTIE	Start Bit Interrupt Enable (0=Disabled; 1=Enabled)
0	UCRXIE	Receive Interrupt Enable (0=Disabled; 1=Enabled)

14.1.3 UART RECEIVE ON THE MSP430FR2355

INTERRUPT SOURCE	INTERRUPT FLAG	INTERRUPT TYPE	VECTOR ADDR	VECTOR LABEL
System Reset	SVSHIFG, PMMRSTIFG, WDTIFG, PMMPORIFG, PMMBORIFG, SYSRSTIV, FLLULPUC	Reset	FFFEh	RESET_VECTOR
System NMI	VMAIFG, JMBINIFG, JMBOUTIFG, CBDIFG, UBDIFG	Non-Maskable	FFFCh	SYSNMI_VECTOR
User NMI	NMIIFG, OFIFG	Non-Maskable	FFFAh	UNMI_VECTOR
Timer0_B3	TB0CCR0 CCIFG0	Maskable	FFF8h	TIMER0_B0_VECTOR
Timer0_B3	TB0CCR1 CCIFG1, TB0CCR2 CCIFG2, TB0IFG (TB0IV)	Maskable	FFF6h	TIMER0_B1_VECTOR
Timer1_B3	TB1CCR0 CCIFG0	Maskable	FFF4h	TIMER1_B0_VECTOR
Timer1_B3	TB1CCR1 CCIFG1, TB1CCR2 CCIF TB1IFG (TB1IV)			
Timer2_B3	TB2CCR0 CCIFG0			
Timer2_B3	TB2CCR1 CCIFG1, TB2CCR2 CCIF TB2IFG (TB2IV)			
Timer3_B7	TB3CCR0 CCIFG0		FECh	TIMER3_B0_VECTOR
Timer3_B7	TB3CCR1 CCIFG1, TB3CCR2 CCIFG2, TB3CCR3 CCIFG3, TB3CCR4 CCIFG4, TB3CCR5 CCIFG5, TB3CCR6 CCIFG6, TB3IFG (TB3IV)	able	FFEAh	TIMER3_B1_VECTOR
RTC counter	RTCIFG	Maskable	FFE8h	RTC_VECTOR
eUSCI_A0 (Rx or Tx)	UCTXCPTIFG, UCSTTIFG, UCRXIFG, UCTXIFG (UART mode) UCRXIFG, UCTXIFG (SPI mode) (UCA0IV))	Maskable	FFE4h	EUSCI_A0_VECTOR
eUSCI_A1 (Rx or Tx)	UCTXCPTIFG, UCSTTIFG, UCRXIFG, UCTXIFG (UART mode) UCRXIFG, UCTXIFG (SPI mode) (UCA0IV)	Maskable	FFE2h	EUSCI_A1_VECTOR

The Rx IRQs share the same eUSCI vector as the Tx IRQs.

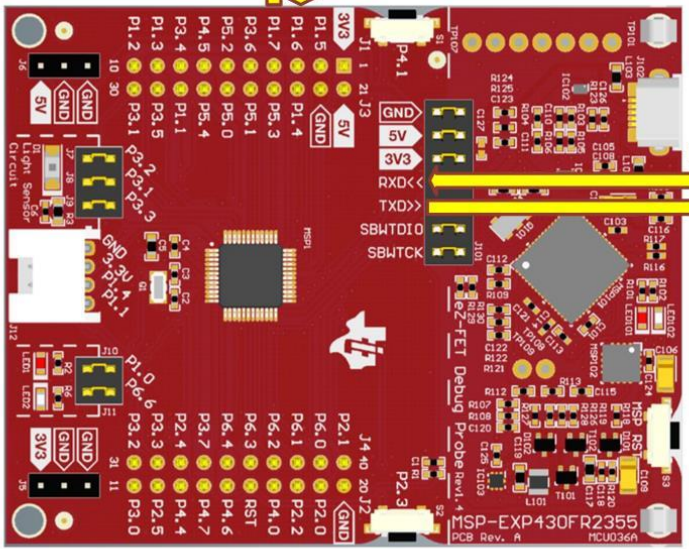
14.1.3 UART RECEIVE ON THE MSP430FR2355

eUSCI_A0 (UART or SPI)
UART Tx (UCA0TXD) is on P1.7
UART Rx (UCA0RXD) is on P1.6

Configuring the P1 pins to be UART Tx/Rx is done in the **P1SEL1:P1SEL0** registers.

P1SEL1(7):P1SEL0(7) = b01 → P1.7 = UCA0TXD
P1SEL1(6):P1SEL0(6) = b01 → P1.6 = UCA0RXD

Tx Rx



Rx
Tx

eUSCI_A1 (UART or SPI)
UART Tx (UCA1TXD) is on P4.3
UART Rx (UCA1RXD) is on P4.2

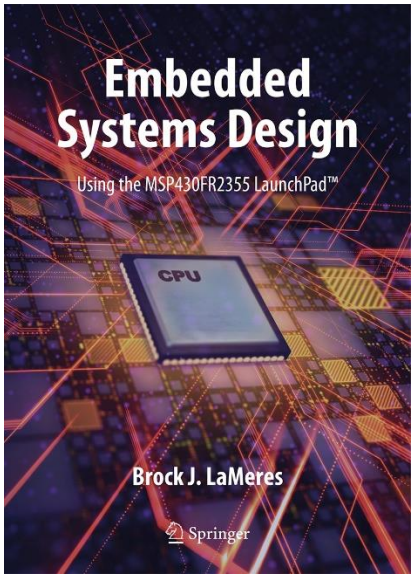
Configuring the P4 pins to be UART Tx/Rx is done in the **P4SEL1:P4SEL0** registers.

P4SEL1(2):P1SEL0(2) = b01
→ P4.2 = UCA1TXD

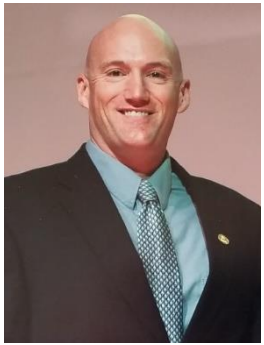
P4SEL1(3):P1SEL0(3) = b01
→ P4.3 = UCA1RXD

CHAPTER 14: SERIAL COMMUNICATIONS IN C

14.1.3 UART RECEIVE ON THE MSP430FR2355 – OVERVIEW

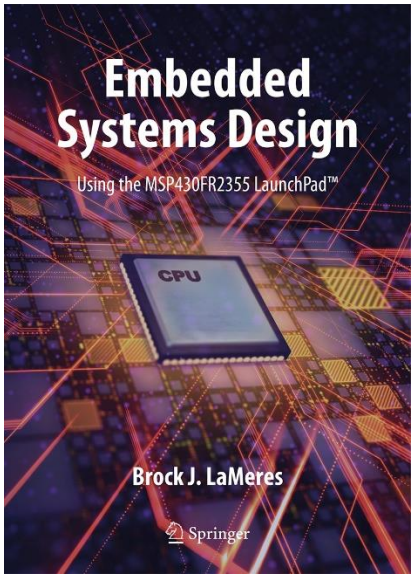


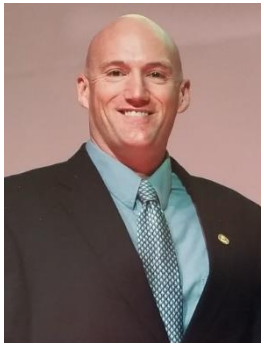
www.youtube.com/c/DigitalLogicProgramming_LaMeres



CHAPTER 14: SERIAL COMMUNICATIONS IN C

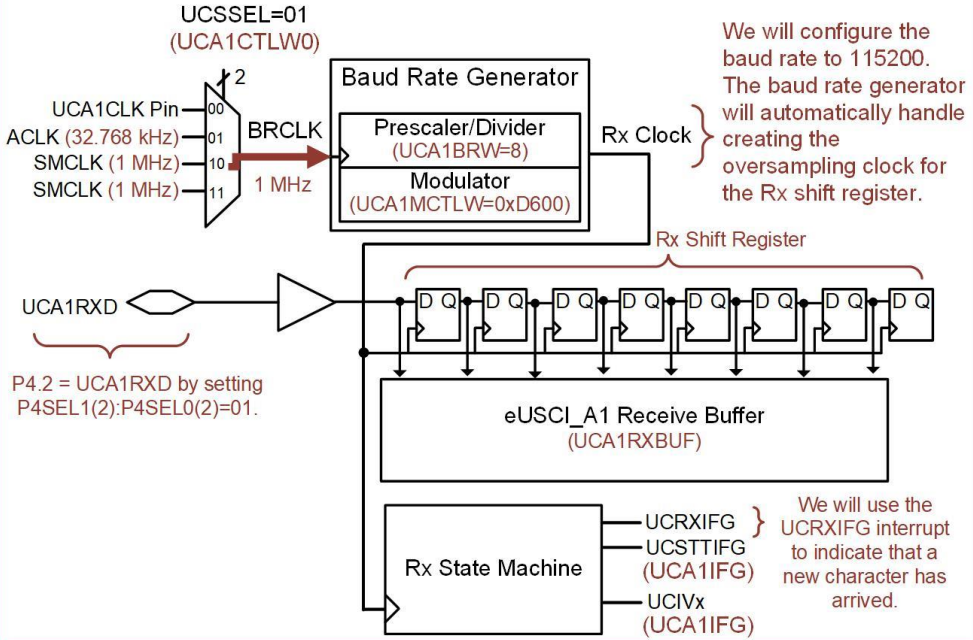
14.1.3 UART RECEIVE ON THE MSP430FR2355 – EXAMPLE: TOGGLING LED1 BASED ON TERMINAL KEYSTROKES





EX: RECEIVING CHARACTERS ON THE EUSCI_A1 UART

Let's design a program that will toglge LED1 on the LaunchPad™ board when the character 't' is received on the eUSCI_A1 UART Rx. We will send characters to the LaunchPad™ board using the terminal within CCS. The terminal handles converting keystrokes into the corresponding ASCII code. We will use the UCRXIFG interrupt to indicate when a new frame has arrived. The following is the eUSCI_A1 setup for this example.



EX: RECEIVING CHARACTERS ON THE EUSCI_A1 UART

Step 1: In CCS, create a new C/C++ Empty Project (with main.c) titled:

C_UART_Rx1_Toggling_LED1

Step 2: Type in the following code in main.c after the statement to stop the watchdog timer.

EX: RECEIVING CHARACTERS ON THE EUSCI_A1 UART

```c
#include <msp430.h>

int main(void)
{
   WDTCTL = WDTPW | WDTHOLD;

   //-- 1. Put eUSCI_A1 into software reset
   UCA1CTLW0 |= UCSWRST;

   //-- 2. Configure eUSCI_A1
   UCA1CTLW0 |= UCSSEL__SMCLK;

   UCA1BRW = 8;
   UCA1MCTLW |= 0xD600;

   //-- 3. Configure Ports
   P4SEL1 &= ~BIT2;
   P4SEL0 |= BIT2;

   P1DIR |= BIT0;
   P1OUT &= ~BIT0;

   PM5CTL0 &= ~LOCKLPM5;

   //-- 4. Take eUSCI_A1 out of software reset
   UCA1CTLW0 &= ~UCSWRST;

   //-- 5. Enable IRQs
   UCA1IE |= UCRXIE;
   __enable_interrupt();

   while(1) {}
   return 0;
}

#pragma vector=EUSCI_A1_VECTOR
__interrupt void EUSCI_A1_RX_ISR(void)
{
    if (UCA1RXBUF == 't')
      {
        P1OUT ^= BIT0;
      }
}
```

The baud rate settings are the same as in the Tx examples.

The UCA1RXD pin shares P4.2. The UART Rx functionality is configured in the P4SEL1:P4SEL0 registers.

Enable the UCA1RXIE interrupt.

The main loop will do nothing. All functionaly will be implemented in the eUSCI_A1 ISR.

This ISR will only run when a new character has been received and is ready to be observed in UCA1RXBUF.

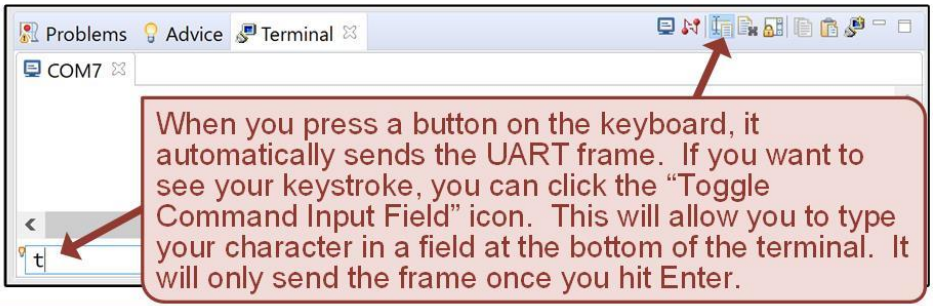
The ISR simply checks if the character received was 't' and if it was, toggles LED1. If it was not 't', it does not toggle LED1.

Reading UCA1RXBUF clears the UCA1RXIFG flag so we don't need to clear it explicitly in the ISR.

EX: RECEIVING CHARACTERS ON THE EUSCI_A1 UART

Step 3: Save and debug your program. Run and then terminate your program. Your program will be running but CCS will not be sharing the USB connection between the debugger and Application UART1.

Step 4: Launch a terminal within CCS and configure it to communicate with the MSP430FR2355 board.



Did it work? Initially you should be able to just press 't' on the keyboard and LED1 will toggle. Make sure the terminal window is active by clicking in it. Next try the "Toggle Command Input Field" option.

EX: RECEIVING CHARACTERS ON THE EUSCI_A1 UART

American Standard Code for Information Interchange (ASCII)

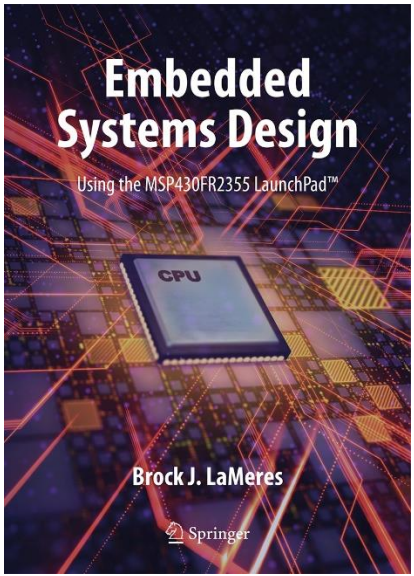
HEX	Char	Description	HEX	Char	Description
0x00	NUL	Null	0x40	@	At sign
0x01	SOH	Start of Header	0x41	A	Capital A
0x02	STX	Start of Text	0x42	B	Capital B
0x03	ETX	End of Text	0x43	C	Capital C
0x04	EOT	End of Transmission	0x44	D	Capital D
0x05	ENQ	Enquiry	0x45	E	Capital E
0x06	ACK	Acknowledge	0x46	F	Capital F
0x07	BEL	Bell	0x47	G	Capital G
0x08	BS	Backspace	0x48	H	Capital H
0x09	HT	Horizontal Tab	0x49	I	Capital I
0x0A	LF	Line Feed	0x4A	J	Capital J
0x0B	VT	Vertical Tab	0x4B	K	Capital K
0x0C	FF	Form Feed	0x4C	L	Capital L
0x0D	CR	Carriage Return	0x4D	M	Capital M
0x0E	SO	Shift Out	0x4E	N	Capital N
0x0F	SI	Shift In	0x4F	O	Capital O
0x10	DLE	Data Link Escape	0x50	P	Capital P
0x11	DC1	Device Control 1	0x51	Q	Capital Q
0x12	DC2	Device Control 2	0x52	R	Capital R
0x13	DC3	Device Control 3	0x53	S	Capital S
0x14	DC4	Device Control 4	0x54	T	Capital T
0x15	NAK	Negative Ack	0x55	U	Capital U
0x16	SYN	Synchronize	0x56	V	Capital V
0x17	ETB	End of Trans Block	0x57	W	Capital W
0x18	CAN	Cancel	0x58	X	Capital X
0x19	EM	End of Medium	0x59	Y	Capital Y
0x1A	SUB	Substitute	0x5A	Z	Capital Z
0x1B	ESC	Escape	0x5B	[	Left Square Bracket
0x1C	FS	File Separator	0x5C	\	Backslash
0x1D	GS	Group Separator	0x5D	]	Right Square Bracket
0x1E	RS	Record Separator	0x5E	^	Caret / Circumflex
0x1F	US	Unit Separator	0x5F	_	Underscore
0x20	space	Space	0x60	`	Grave / Accent
0x21	!	Exclamation Mark	0x61	a	Small a
0x22	"	Double Quote	0x62	b	Small b
0x23	#	Number	0x63	c	Small c
0x24	$	Dollar sign	0x64	d	Small d
0x25	%	Percent	0x65	e	Small e
0x26	&	Ampersand	0x66	f	Small f
0x27	'	Single Quote	0x67	g	Small g
0x28	(	Left Parenthesis	0x68	h	Small h
0x29	)	Right Parenthesis	0x69	i	Small i
0x2A	*	Asterisk	0x6A	j	Small j
0x2B	+	Plus	0x6B	k	Small k
0x2C	,	Comma	0x6C	l	Small l
0x2D	-	Minus	0x6D	m	Small m
0x2E	.	Period	0x6E	n	Small n
0x2F	/	Slash	0x6F	o	Small o
0x30	0	Zero	0x70	p	Small p
0x31	1	One	0x71	q	Small q
0x32	2	Two	0x72	r	Small r
0x33	3	Three	0x73	s	Small s
0x34	4	Four	0x74	t	Small t
0x35	5	Five	0x75	u	Small u
0x36	6	Six	0x76	v	Small v
0x37	7	Seven	0x77	w	Small w
0x38	8	Eight	0x78	x	Small x
0x39	9	Nine	0x79	y	Small y
0x3A	:	Colon	0x7A	z	Small z
0x3B	;	Semicolon	0x7B	{	Left Curly Bracket
0x3C	<	Less Than	0x7C	|	Vertical Bar
0x3D	=	Equality Sign	0x7D	}	Right Curly Bracket
0x3E	>	Greater Than	0x7E	~	Tilde
0x3F	?	Question Mark	0x7F	DEL	Delete

HEX	Char	Description
0x40	@	At sign
0x41	A	Capital A
0x42	B	Capital B
0x43	C	Capital C
0x44	D	Capital D
0x45	E	Capital E
0x46	F	Capital F
0x47	G	Capital G

Can you see these ASCII codes in the debugger?

CHAPTER 14: SERIAL COMMUNICATIONS IN C

14.1.3 UART RECEIVE ON THE MSP430FR2355 – EXAMPLE: TOGGLING LED1 BASED ON TERMINAL KEYSTROKES



www.youtube.com/c/DigitalLogicProgramming_LaMeres