Lecture 11: Serial Communication 1

ME/AE 6705

Introduction to Mechatronics

Dr. Jonathan Rogers

Georgia Tech

Lesson Objectives

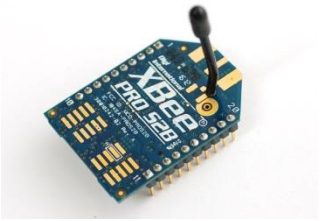
- Understand format and terminology of serial communication
- Be able to explain differences between RS-232 and TTL data formats
- Be able to configure UART interface on MSPM0
- Be able to implement interrupt-based serial communication at proper rates

Interdevice Communication

- Need for MCU to communicate with external devices arises often in Mechatronics
- Oftentimes we need MCU to share digital data with another device. Examples:
  - MCU receives data from GPS receiver
  - MCU is programmed by computer
  - MCU transmits data to computer (data acquisition)
  - MCU communicates wirelessly with other MCU's

ublox GPS receiver

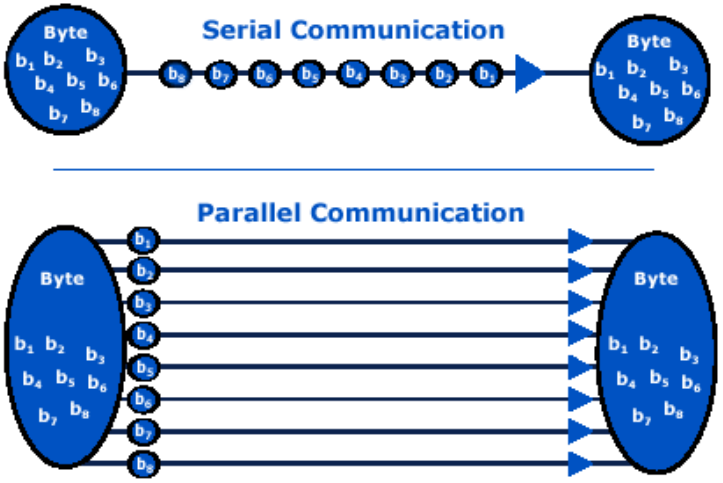
Pic Programmer



XBee Wireless Chip

Serial vs Parallel Communication

- When two devices communicate binary data, there are two ways to implement it:
  - Parallel – multiple bits are sent simultaneously
  - Serial – one bit sent at a time



- Parallel communication requires more TX/RX lines (wires)
- 2-way serial can be implemented with just 3 wires
- Parallel communication used to be popular because it provided higher communication bandwidth

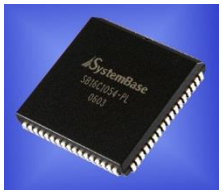
Now that processor speeds are so fast, simplicity of serial has outweighed bandwidth penalty. Now serial is used almost exclusively.

Synchronous vs Asynchronous Communication

- Likewise, interdevice communication can occur synchronously or asynchronously
- Synchronous:
  - Data is sent in continuous stream at constant rate
  - Clocks in transmitting and receiving devices must run at exactly the same rate (be synchronized)
- Asynchronous:
  - Data sent as needed, not in continuous stream
  - Clocks can be running at different rates, but devices just agree on data rate
  - Start and stop bits used to synchronize
    - *This is overhead incurred for running asynchronously*

Modern MCU Communication

- Digital communication to and from modern MCU's is commonly done using *asynchronous serial* interfaces
  - For instance, Universal Asynchronous Receiver Transmitter (UART) hardware device
- To use UART successfully, must know:
  1. Serial communication protocol (RS-232 and TTL serial)
  2. How to configure UART successfully (on both devices)



SB16C1054 Quad UART IC

Georgia Tech

UART Serial Communication

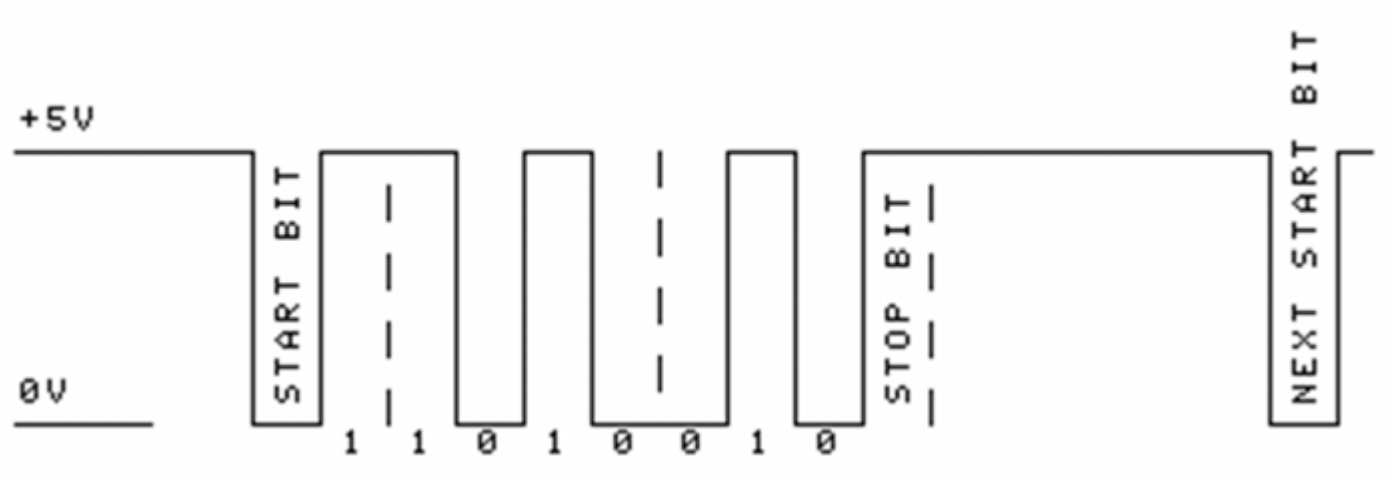
- UART Serial transmission involves sending one bit at a time
  - For 1-way communications, only two wires are needed:
    - *Signal, Ground*
  - For 2-way communications, three wires are needed:
    - *Signal out, Signal in, Ground*
- To send a byte of data, we must send 8 bits individually, plus a start and a stop bit

  → Total of 10 bits for each byte of data we send!

Georgia Tech

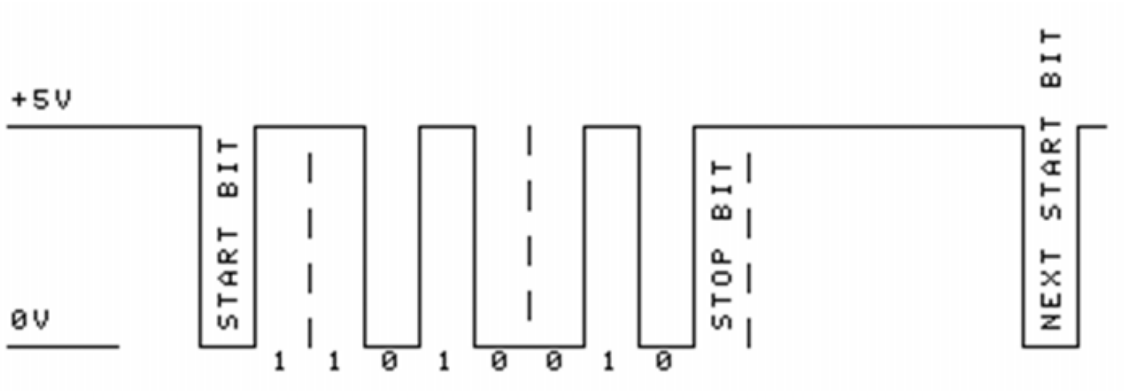
TTL Serial Communication

- TTL = Transistor-transistor logic
  - Basically means 3.3V (or 5V) is high, 0V is low
- 1 byte transmitted at a time (called a “frame”)
  - LSB transmitted first, MSB transmitted last
- Transmission of 01001011 (75 decimal) using 5V TTL levels



Serial Communication

- Start bit: Transmission line is at 5V at idle
  - To start transmission, line goes low (start bit)
- Data bits: 8 data bits sent at one time, LSB first
- Stop bit: Line goes high after 8th data bit
  - This signifies end of current frame
  - Process is repeated for next frame



Georgia Tech

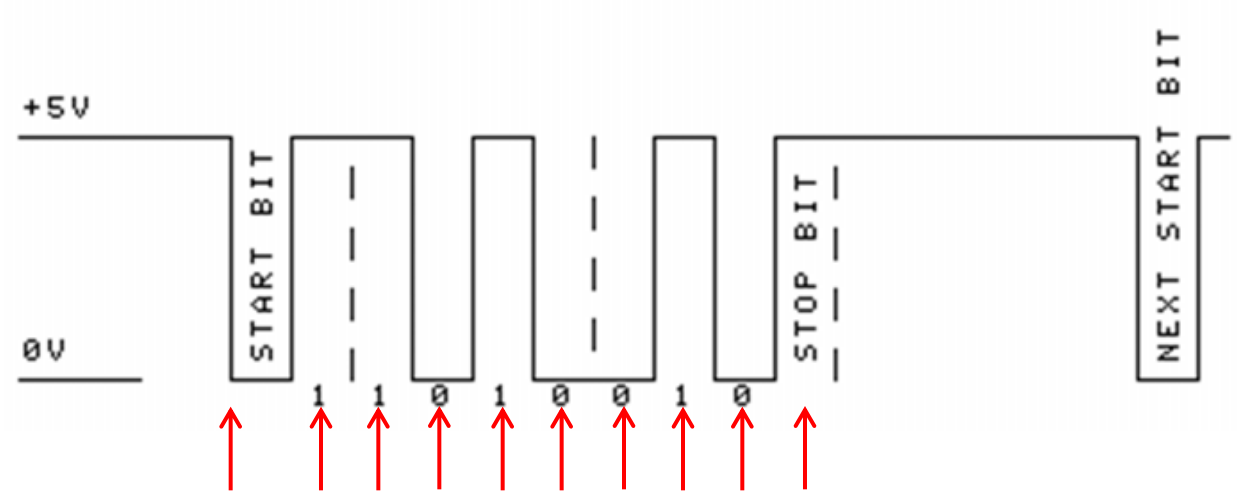
Baud Rate

- Total number of bits transmitted per second is called the **baud rate**
- The bit time T is 1/(baud rate), signifying how long it takes to transmit a single bit
- Baud rate is set during serial comms configuration
  - Both devices must use the same baud rate
- Baud rate tradeoffs:
  - High baud rates allow higher communication speed
  - Low baud rates ensure higher accuracy

Bytes per sec = (Baud rate) / 10

Sampling the Serial Frame

- Consider previous example:



  - Once start bit is detected (line goes low), receiver waits 1.5 x T and samples line 9 times at T sec intervals to read data and stop bit (T is bit time)

Baud Rate Examples

- How long does it take to transmit each bit if the baud rate is 9600? How many bytes are transmitted each second?

- If your timing is off by 4 µs each time you read a bit, will the data be read correctly at:
  - 9600 baud?
  - 38400 baud?

Georgia Tech

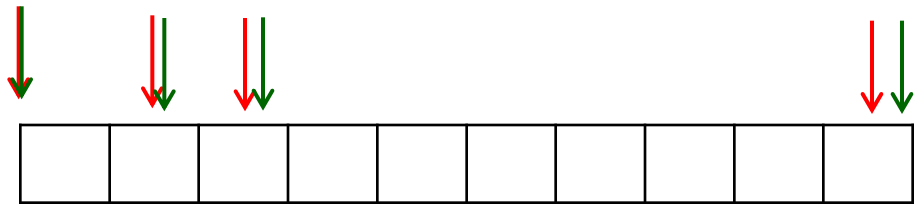
Baud Rate Examples

- How long does it take to transmit each bit if the baud rate is 9600? How many bytes are transmitted each second?

T = 1/9600 = 104.2 μs

Bytes per sec = 9600/10 = 960

- If your timing is off by 4 μs each time you read a bit, will the data be read correctly at:
  - 9600 baud? T = 1/9600 = 104.2 μs

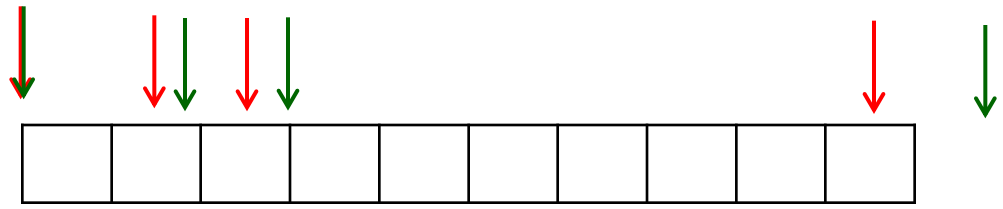


- By last bit, we will be 9 x 4 = 36 μs off, which is still well within half of our bit time
- So data will be read correctly

Baud Rate Examples

- How long does it take to transmit each bit if the baud rate is 9600? How many bytes are transmitted each second?

- If your timing is off by 4 μs each time you read a bit, will the data be read correctly at:
  - 38400 baud? T = 1/38400 = 26.0 μs



- By last bit, we will be 9 x 4 = 36 μs off, which way longer than half our bit time
- So data will not be read correctly

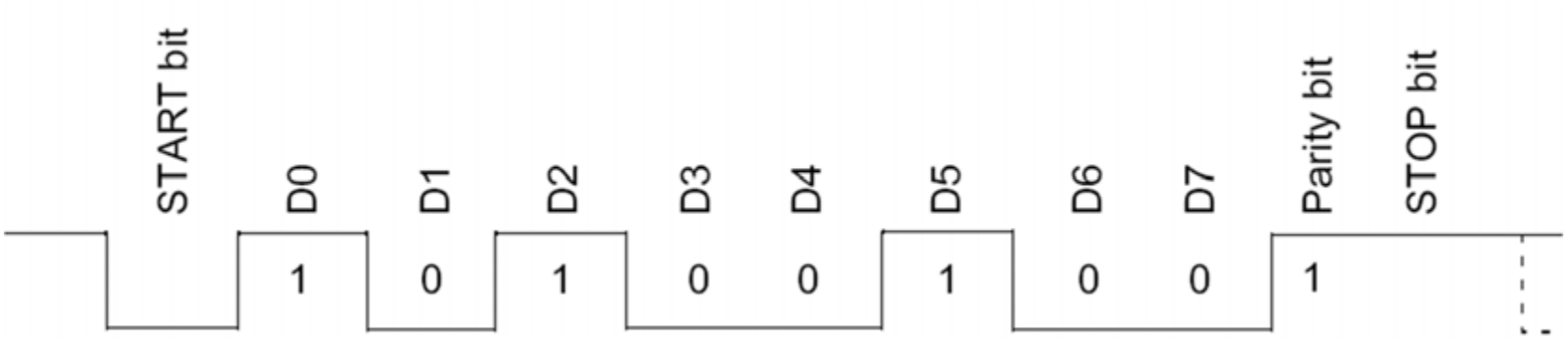
Parity Bits

- Some protocols use a parity bit for error checking
  - Added before the stop bit
  - One parity bit means we need 11 bits to transmit a byte
- Even parity:
  - Ones in data and parity bit sum to even
- Odd parity:
  - Ones in data and parity bit sum to odd
- Use of parity bit, and even vs odd parity, is set during serial comms configuration

Georgia Tech

Parity Bits

- Even parity example: Transmit 37 decimal with an even parity bit



  - 37 in binary is 00100101. Sum of ones is 3, so parity bit must be 1 to be even

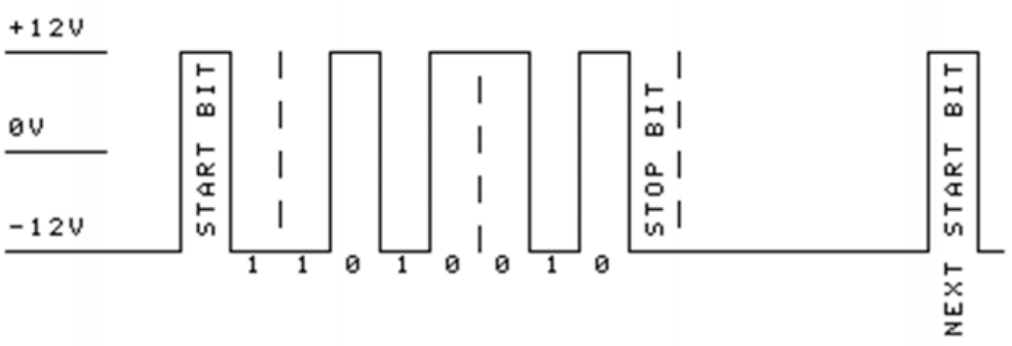
Georgia Tech

RS-232

- RS-232 is a communication protocol used by computers (not by microcontrollers)
- Same basic protocol except:
  - +12 V is considered “low”
  - -12 V is considered “high”
  - 0 V is ground reference

Typical values used in PC’s, but can vary between 5-15V

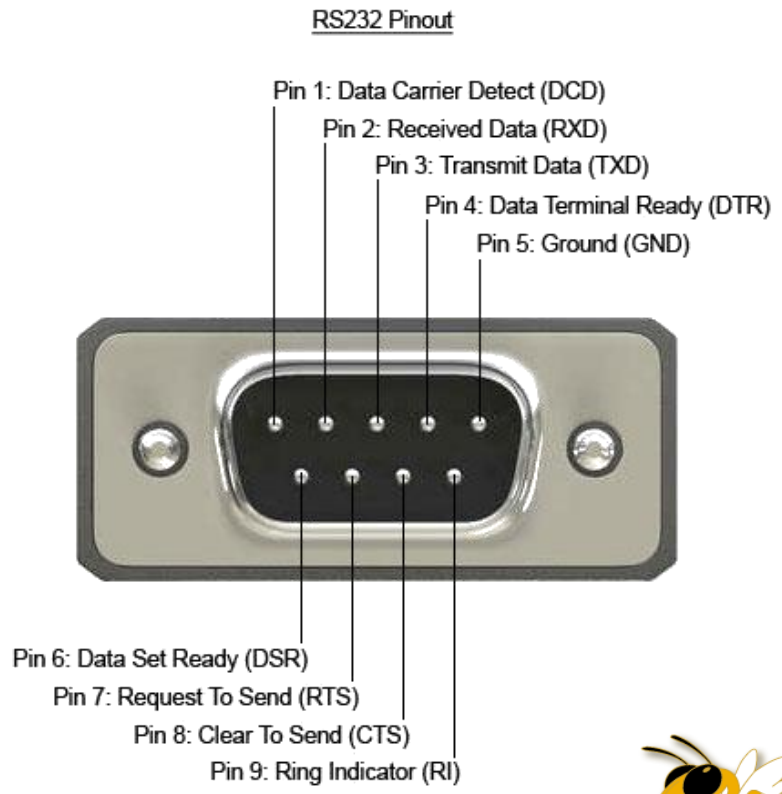
- Transmission of decimal 75 using RS-232:



Georgia Tech

RS-232

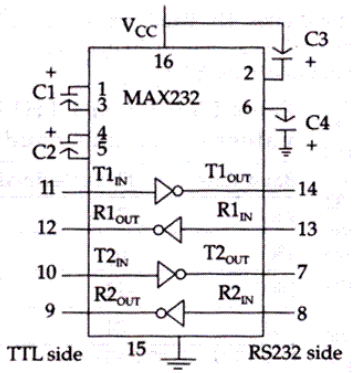
- Older computers had RS-232 port which used DB9 connectors
  - This was before USB
- 9 total wires, but for simple RS-232 communication only 3 wires are needed
  - Pin 2 (receive)
  - Pin 3 (transmit)
  - Pin 5 (common ground)



Georgia Tech

Level Converters

- MCU's cannot support voltages above 5V (or for MSPM0, 3.3 V)
- When interfacing MCU's with computers, we usually use **MAX232 level converter chip**
  - Converts between -12/+12V RS-232 signals and 0-3.3V or 0-5V signals which are used by MCU
  - Also inverts so that -12V→3.3V and +12V →0V, and vice versa



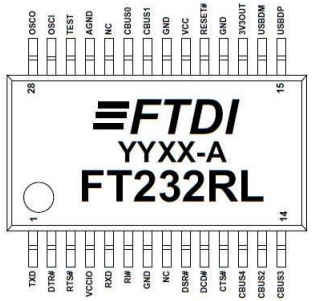
Cost ~$2

Georgia Tech

USB to TTL Serial Interface

- USB (universal serial bus) is a serial protocol that is used widely
- To convert USB signals to TTL serial signals, an **FT232R chip** can be used
  - Entire conversion process to USB protocol handled on chip, no programming required
  - Versions for 0-5V, 0-3.3V, etc.
  - Can handle 300 baud to 3 Mbaud

Cost ~$4

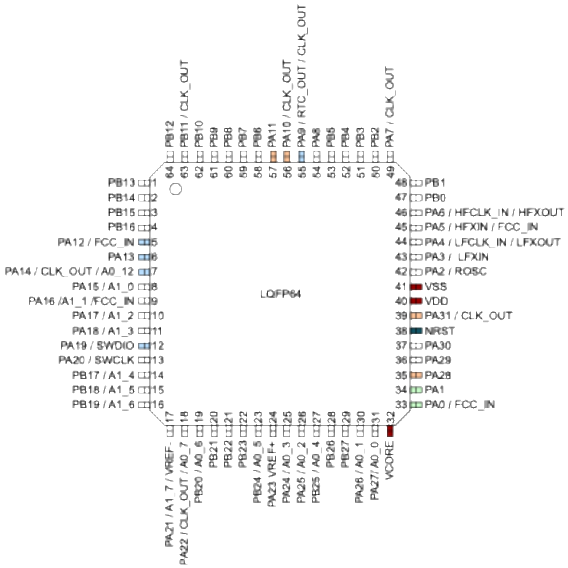


Georgia Tech

UART on MSPM0

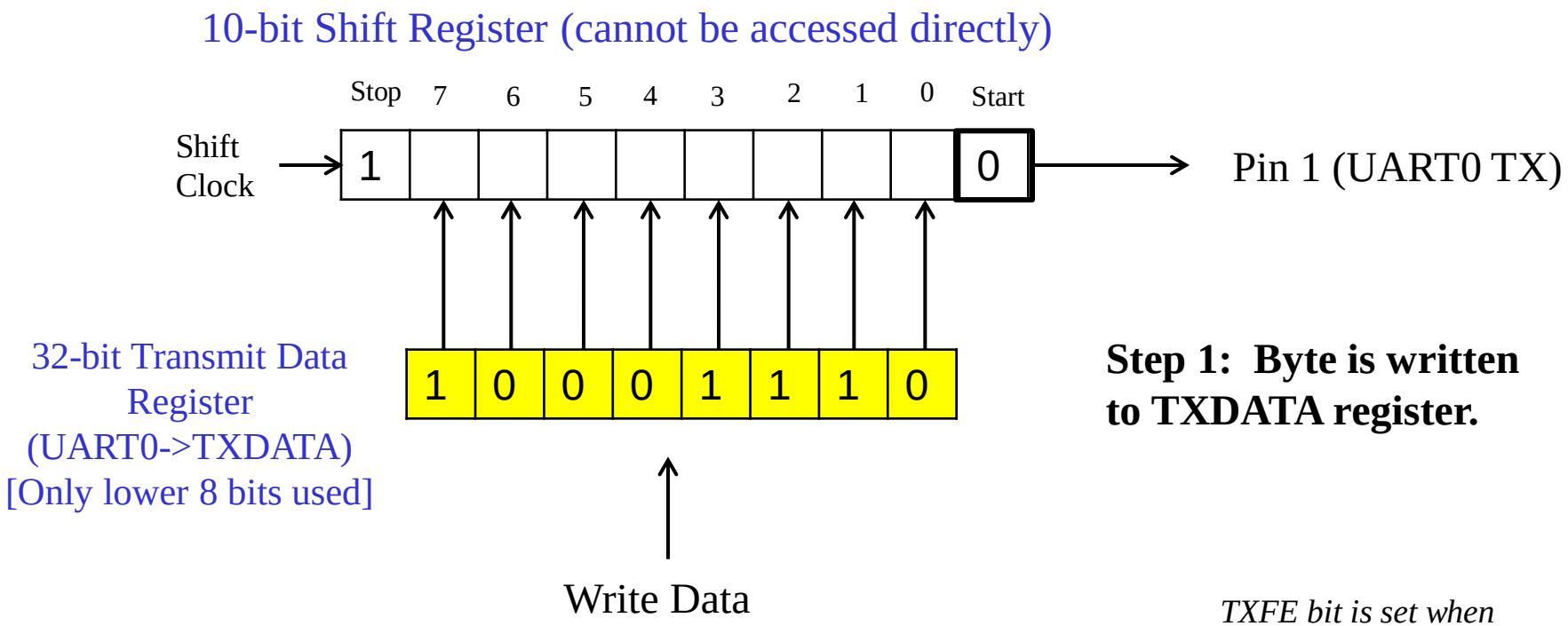
- UART is physical hardware device that implements serial protocols (TTL) described above
  - You can buy UART IC's, but they are already integrated into many MCU's
  - MSPM0 has four UARTs. From datasheet, we can find:

UART	TX/RX	Available on Pins
UART 0	TX	1, 3, 12, 21
	RX	2, 6, 13, 22
UART 1	TX	17, 19, 23, 39
	RX	18, 20, 24, 40
UART 2	TX	32, 43, 46, 53
	RX	33, 44, 47, 54
UART 3	TX	15, 29, 36, 59
	RX	16, 30, 35, 55



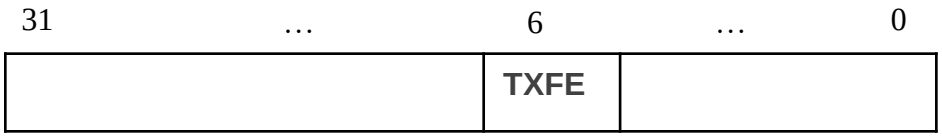
UART Transmit

- On MSPM0 UART data is transmitted 1 byte at a time



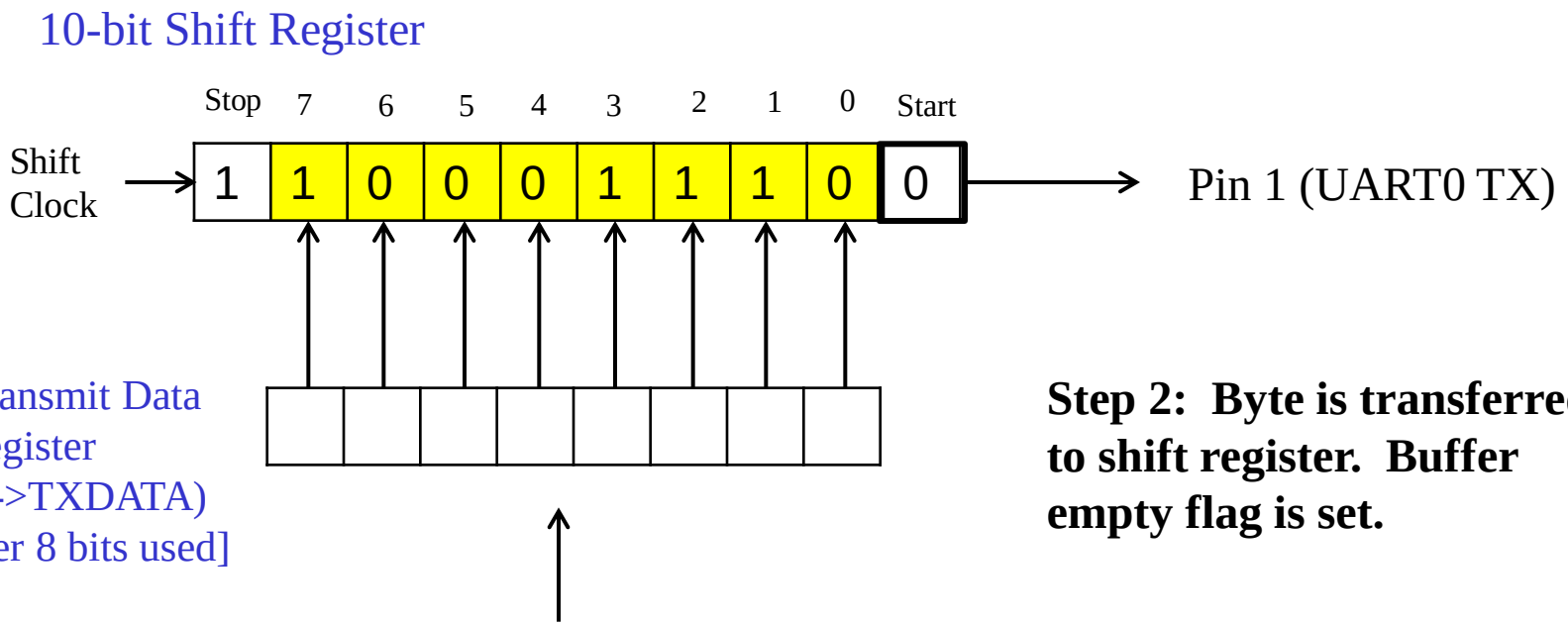
TXFE bit is set when TXDATA is empty

UART0->STAT Register (32-bit)



UART Transmit

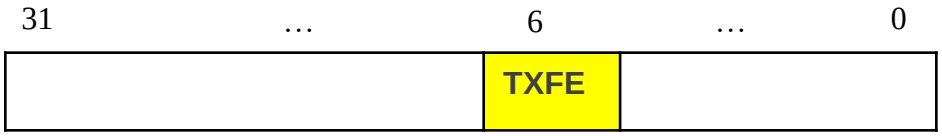
- On MSPM0 UART data is transmitted 1 byte at a time



Step 2: Byte is transferred to shift register. Buffer empty flag is set.

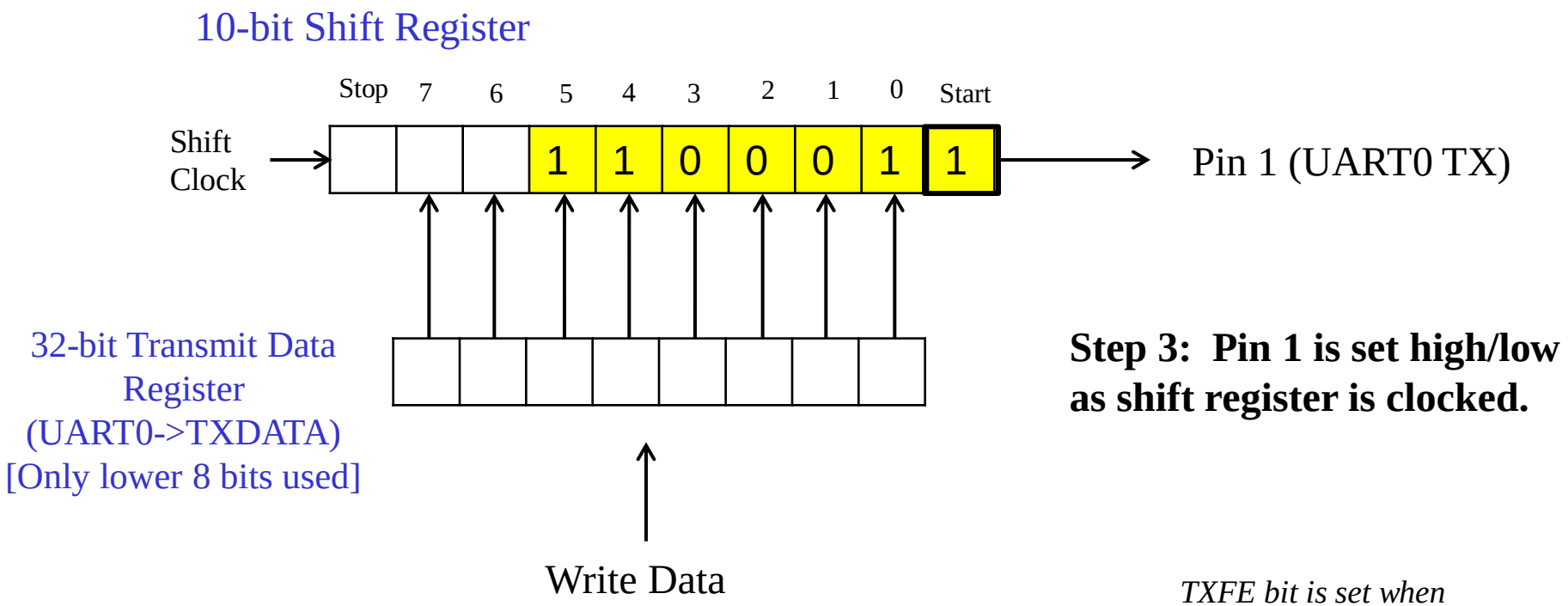
TXFE bit is set when TXDATA is empty

UART0->STAT Register (32-bit)

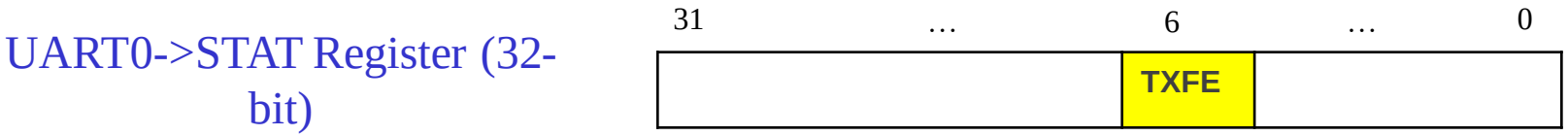


UART Transmit

- On MSPM0 UART data is transmitted 1 byte at a time

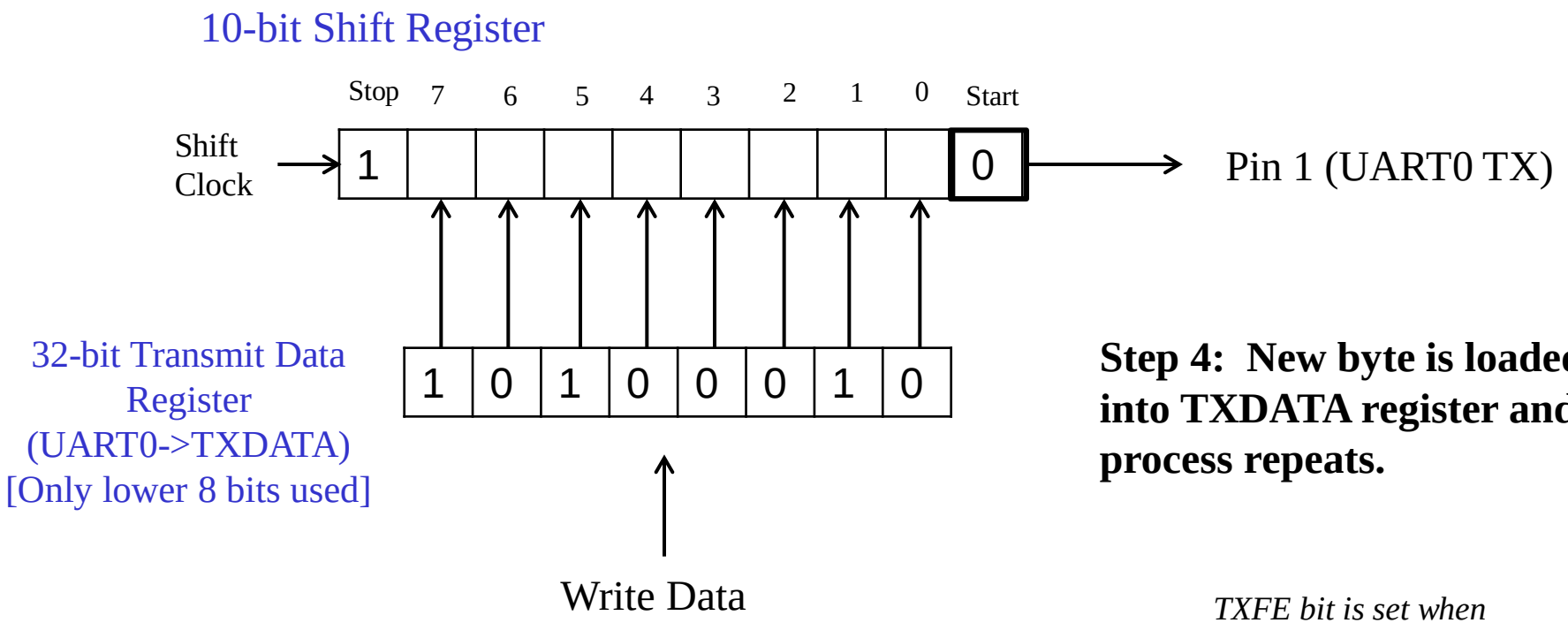


TXFE bit is set when TXDATA is empty

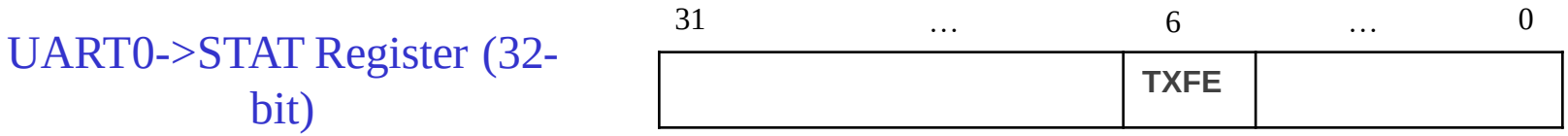


UART Transmit

- On MSPM0 UART data is transmitted 1 byte at a time



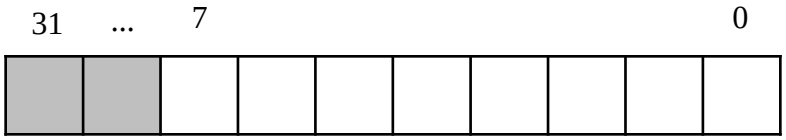
TXFE bit is set when TXDATA is empty



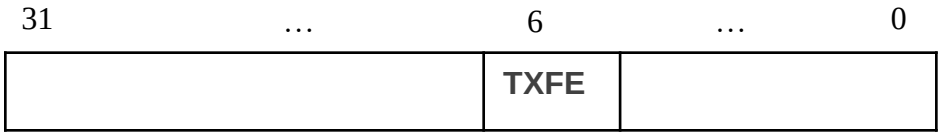
UART Transmit

- Suppose you want to send more than 1 byte
  - Then you must break your data into 1 byte chunks, and write 1 byte at a time
- Note: You must check the TXFE bit in UARTx->STAT before writing a byte
  - If TXFE = 0, then there is currently data in the register waiting to be written
  - If you write something else, it will be lost!

32-bit Transmit Data Register (UARTx->TXDATA)



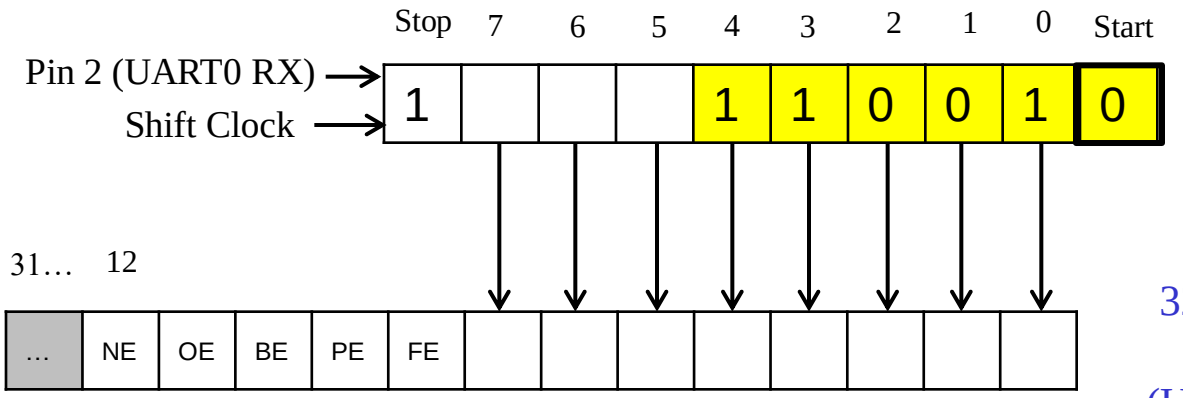
UARTx->STAT Register (32-bit)



UART Receive

- On MSPM0 UART data is received 1 byte at a time

10-bit Shift Register

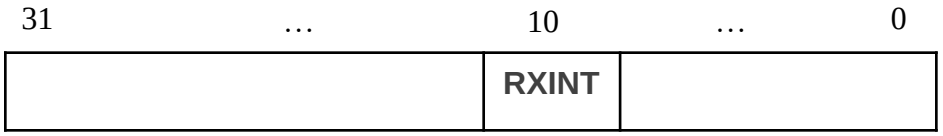


Step 1: Start bit received and data read one bit at a time.

32-bit Receive Data Register (UART0->RXDATA)

RXINT bit is set when data has been received and is ready to read.

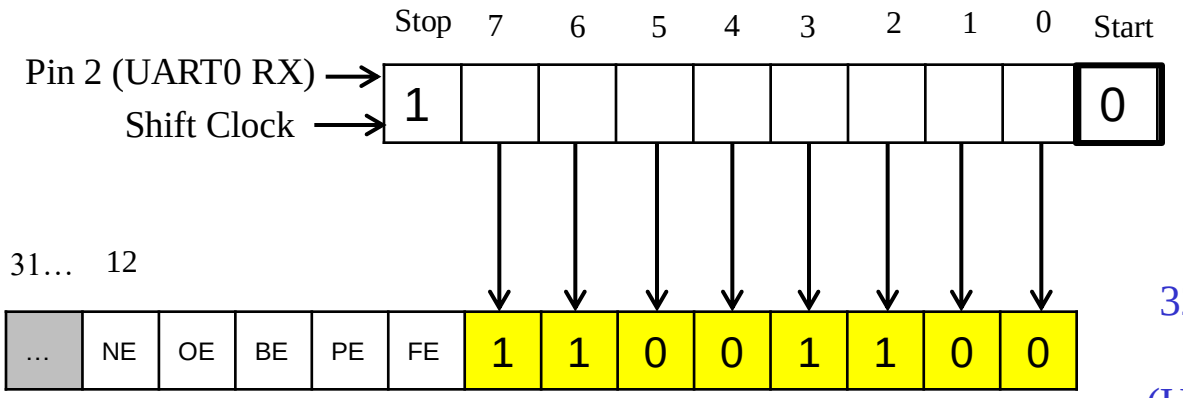
UART0->CPU_INT.RIS Register (32-bit)



UART Receive

- On MSPM0 UART data is received 1 byte at a time

10-bit Shift Register

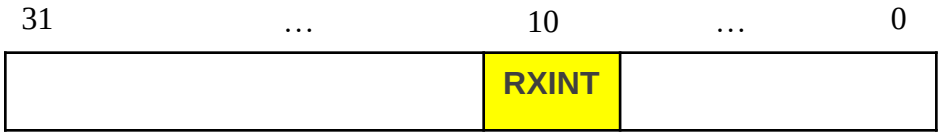


Step 2: After stop bit received, byte is transferred to receive buffer register. RXINT flag is set.

32-bit Receive Data Register (UART0->RXDATA)

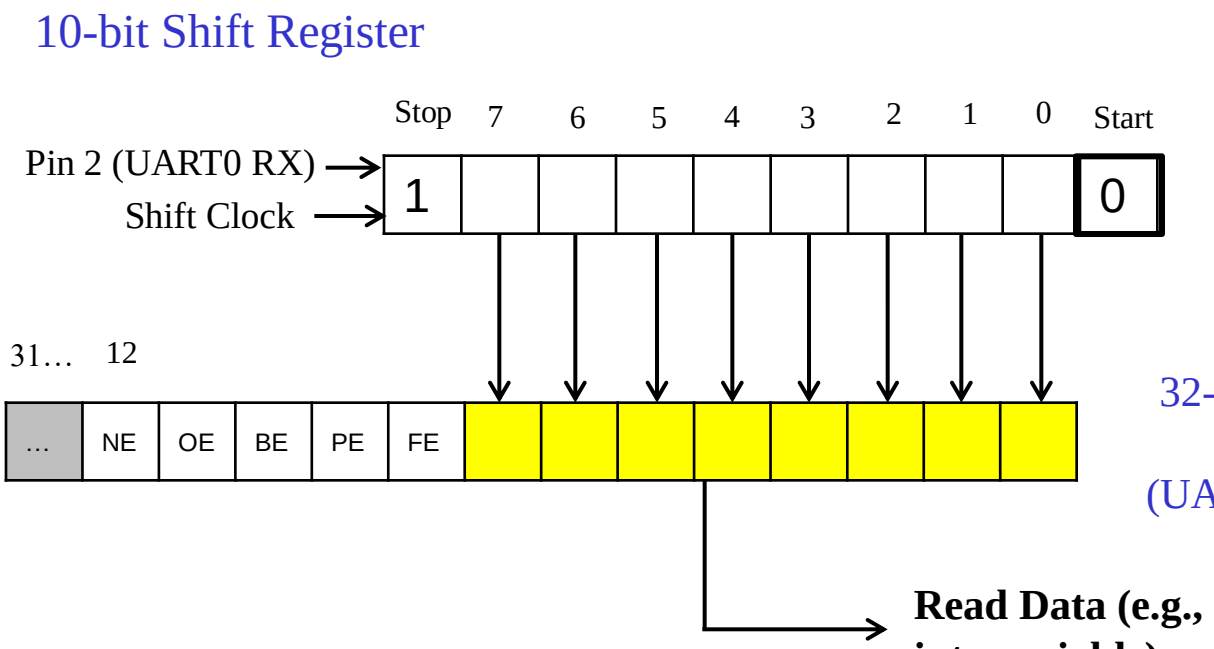
RXINT bit is set when data has been received and is ready to read.

UART0->CPU_INT.RIS Register (32-bit)



UART Receive

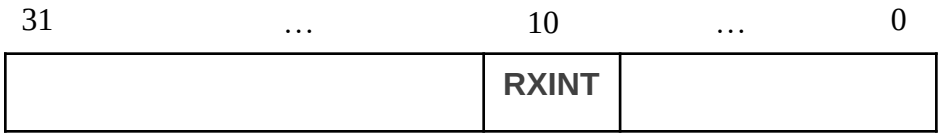
- On MSPM0 UART data is received 1 byte at a time



Step 3: Data is read from the receive buffer register, and the RXINT flag is automatically cleared. Process then repeats.

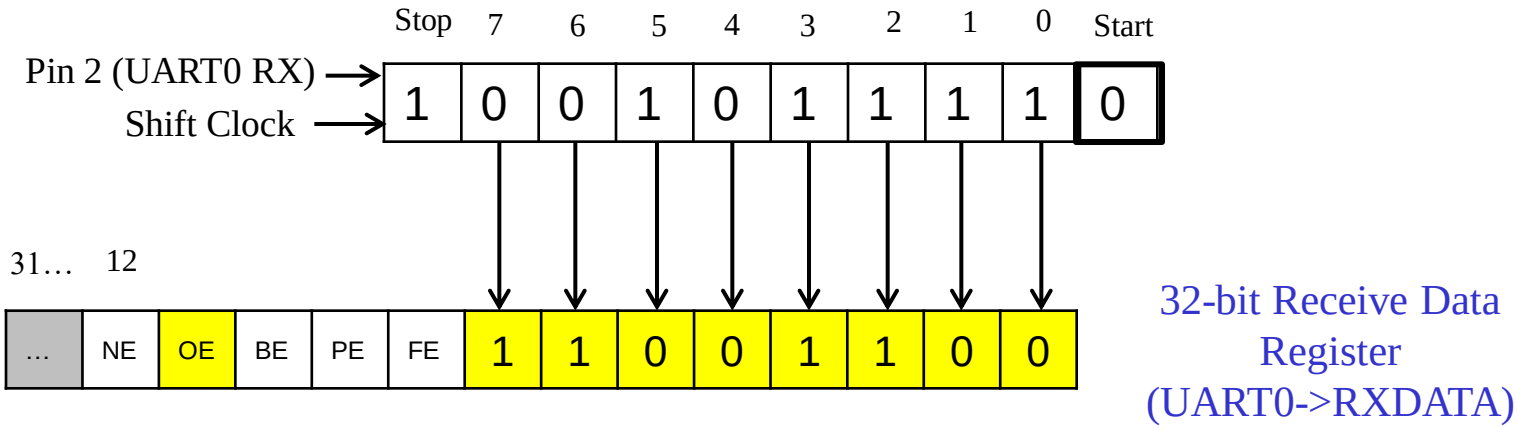
RXINT bit is set when data has been received and is ready to read.

UART0->CPU_INT.RIS Register (32-bit)



UART Receive – Error Conditions

- **Overflow Error** - Suppose both receive buffer and shift register are full when another byte comes in
  - *This means you have not read the data in time – your program is not keeping up*
  - *This is known as an overflow error (OE) – will set OE bit*

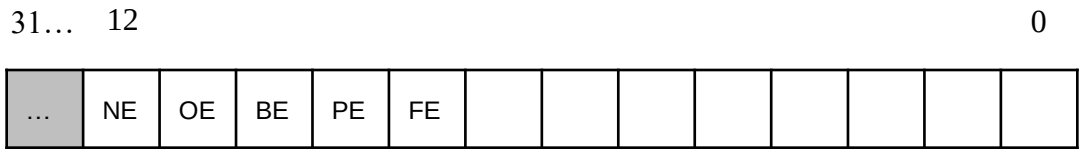


Overflow bit will remain high until overflow error clear bit is set in UART EVENT ICLR register.

Must make sure you read data fast enough for your chosen baud rate.

UART Receive – Error Conditions

- Break error: Set when input is held low for more than a frame (BE bit set)
  - Indicates line has gone idle
- Parity error: Set when parity error detected (PE bit set)
  - Most systems do not implement parity bit because serial comms are reliable enough
- Framing error: Set when stop bit is incorrect (FE bit set)
  - Most likely means there is a mismatch in baud rate

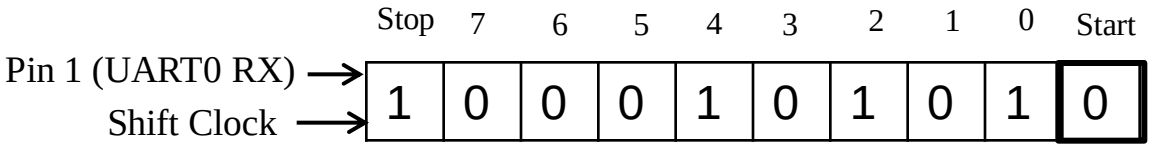


32-bit Receive Data Register (UART0->RXDATA)

Georgia Tech

Oversampling

- The MSPM0 UART implements something called **oversampling**
  - Enabled by setting bits 15 and 16 of UARTx->CTL0 register (where x = 0, 1, 2, 3)
- Oversampling means that shift register operates at 1/16 rate at which input line (UARTx RX) sampled
  - Allows UART to sample input signal multiple times for each bit
  - Sampling clock runs 16 times faster than shift clock (which runs at baud rate)



Selecting the Baud Rate

- When configuring the baud rate, we set UARTx->IBRD and UARTx->FBRD registers
  - BRD holds baud rate divisor
  - FBRD holds fractional baud rate divisor

$$\textbf{Baud Rate} = \frac{\text{Selected Clock Frequency}}{\text{Oversampling Rate} \times (\textbf{IBRD} + \textbf{64} \times \textbf{FBRD})}$$

  - We can select a clock frequency to operate at (will discuss two lectures from now)
  - Then just set IBRD and FBRD registers to values we want

Georgia Tech

Selecting the Baud Rate

- Example: Suppose we have a clock frequency of 40 MHz and we want a baud rate of 19200 bits/sec
  - This is one of standard baud rates
  - What should our dividers be?

Selecting the Baud Rate

- Example: Suppose we have a clock frequency of 40 MHz and we want a baud rate of 19200 bits/sec
  - This is one of standard baud rates
  - What should our dividers be?

$$\mathbf{IBRD} + \mathbf{64} \times \mathbf{FBRD} = \frac{\text{Selected Clock Frequency}}{\text{Oversampling Rate} \times \mathbf{(Baud\ Rate)}} \frac{40\ \text{MHz}}{16 \times 19200\ \text{bit/s}} = 130.208333$$

$$IBRD = 130$$

$$FBRD = INT\big((64 \times .208333) + 0.5\big) = INT(13.8333) = 13$$

Added to minimize rounding error when rounding to int

Georgia Tech

Selecting the Baud Rate

- Example: Suppose we have a clock frequency of 40 MHz and we want a baud rate of 19200 bits/sec
  - This is one of standard baud rates
  - What should our dividers be?

```
UART0->IBRD = 130 ;
UART0->FBRD = 13 ;
```

- When we write above values, actual baud rate will be 19200.77 bit/s.
- As a rule of thumb, baud rates between transmitter and receiver must match within 5%. This one is well within that so we are okay.

Georgia Tech

Configuring MSPM0 UART

- Driverlib provides easy mechanism to configure UART
- Primary function to configure UART:

```
void DL_UART_init(UART_Regs * uart, DL_UART_Config * config)
```

Argument 1: UART module

```
UART0
UART1
UART2
UART3
```

Defined in dl_uart.h

Argument 2: Configuration Struct

```
typedef struct {
{
    DL_UART_MODE mode ;
    DL_UART_DIRECTION direction ;
    DL_UART_FLOW_CONTROL flowControl ;
    DL_UART_PARITY parity ;
    DL_UART_WORD_LENGTH wordLength ;
    DL_UART_STOP_BITS stopBits
} DL_UART_Config;
```

Configuring MSPM0 UART

- Enable module
  - After configuring using initModule, UART must be enabled before it is active

```
void DL_UART_enable(UART_Regs *uart)
```

- Enabling interrupts
  - You can enable transmit interrupts, receive interrupts, or both
  - This will trigger interrupt when RXIFG or TXIFG bit is set

```
void DL_UART_enableInterrupt( UART_Regs *uart, uint32_t interruptMask)
```

Argument 2 (mask) can be:

```
DL_UART_INTERRUPT_TX
DL_UART_INTERRUPT_RX
(or both bitwise or'd together)
```

Configuring MSPM0 UART

- Turn power on to UART peripheral

```
void DL_UART_enablePower(UART_Regs *uart)
```

- Set clock configuration

```
void DL_UART_setClockConfig (UART_Regs *uart, DL_UART_ClockConfig * config)
```

```
typedef struct {
{
   DL_UART_CLOCK clockSel
   DL_UART_CLOCK_DIVIDE_RATIO divideRatio
} DL_UART_ClockConfig;
```

Georgia Tech

Configuring MSPM0 UART

- Setting the baud rate
  - First set the oversampling rate among the following options: DL_UART_OVERSAMPLING_RATE_16X, DL_UART_OVERSAMPLING_RATE_8X, DL_UART_OVERSAMPLING_RATE_3X

```
void DL_UART_setOversampling (UART_Regs *uart, DL_UART_OVERSAMPLING_RATE rate)
```

  - Then set the constants you want written to the IBRD and FBRD registers

```
void DL_UART_setBaudRateDivisor(UART_Regs *uart, uint32_t integerDivisor, uint32_t fractionalDivisor)
```

Georgia Tech

Using MSPM0 UART

- Receive data:

```
uint8_t  DL_UART_receiveData (UART_Regs *uart)
```

Returns 8-bit value in UARTx RXDATA register, cast as a uint8_t

- Transmit data:

```
void DL_UART_transmitData (UART_Regs *uart,
                           uint8_t data)
```

Transmits 8-bit value in in second function argument (places it in UARTx TXDATA register)

  - Note: You should always check TXFE flag before sending data.
  - If zero, there is still data in TXDATA waiting to be sent, and if you send another byte it will be lost.

ASCII

- When we send data to PC, we usually want it to be human-readable
- Suppose we want to send the number “4” to PC
  - Two ways of doing it:

Send Data as Binary Value

```
DL_UART_transmitData(UART0, 0x02) ;
```

- If this is printed to screen, it will not look like anything
- The ASCII character associated with 0x02 is special “end of transmission” character

Send Data as ASCII Value

```
char number = ‘4’ ;
DL_UART_transmitData(UART0, number) ;
```

- This will print to the screen successfully
- When we initialize “number” as ‘4’, it writes ASCII value associated with 4 (0x34) to variable

PuTTY and Serial Port Monitors

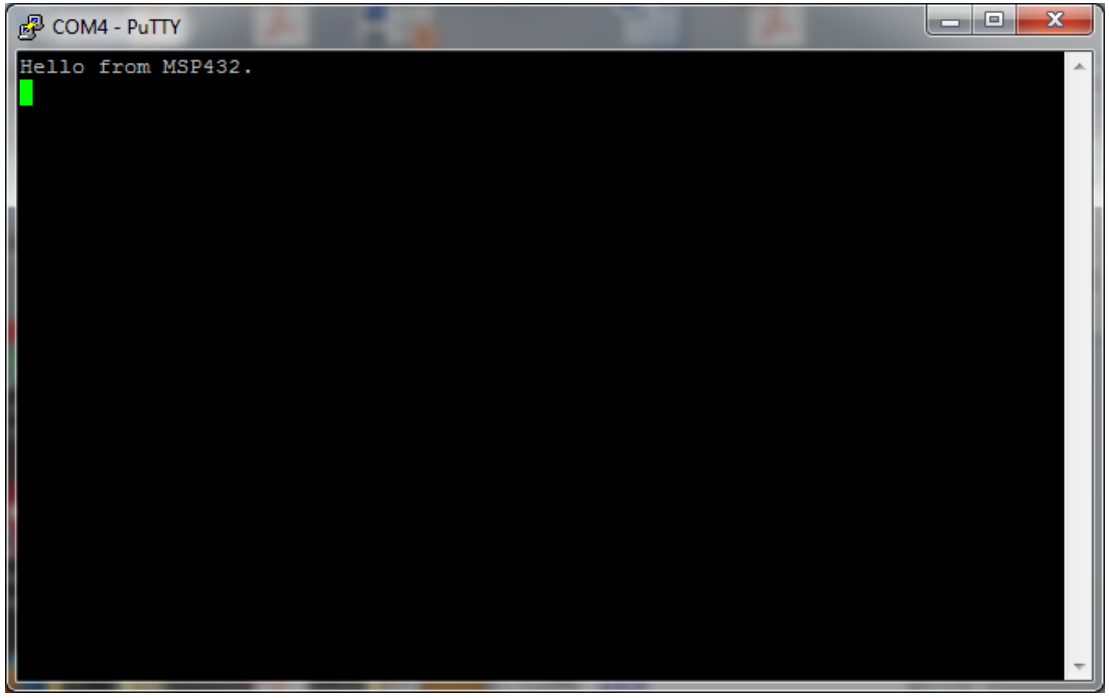
- Best way to debug serial communications code is with serial port monitor
  - Displays serial output to screen so you can see what is going on
  - Example: http://www.serial-port-monitor.com/
- PuTTY is free program that effectively turns your computer's USB port into a serial port
  - Typing data into PuTTY window, sends this data outbound on serial line

Georgia Tech

PuTTY

- PuTTY is free to download:

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html



Georgia Tech