香港中文大學
The Chinese University of Hong Kong

CENG2400 Embedded System Design

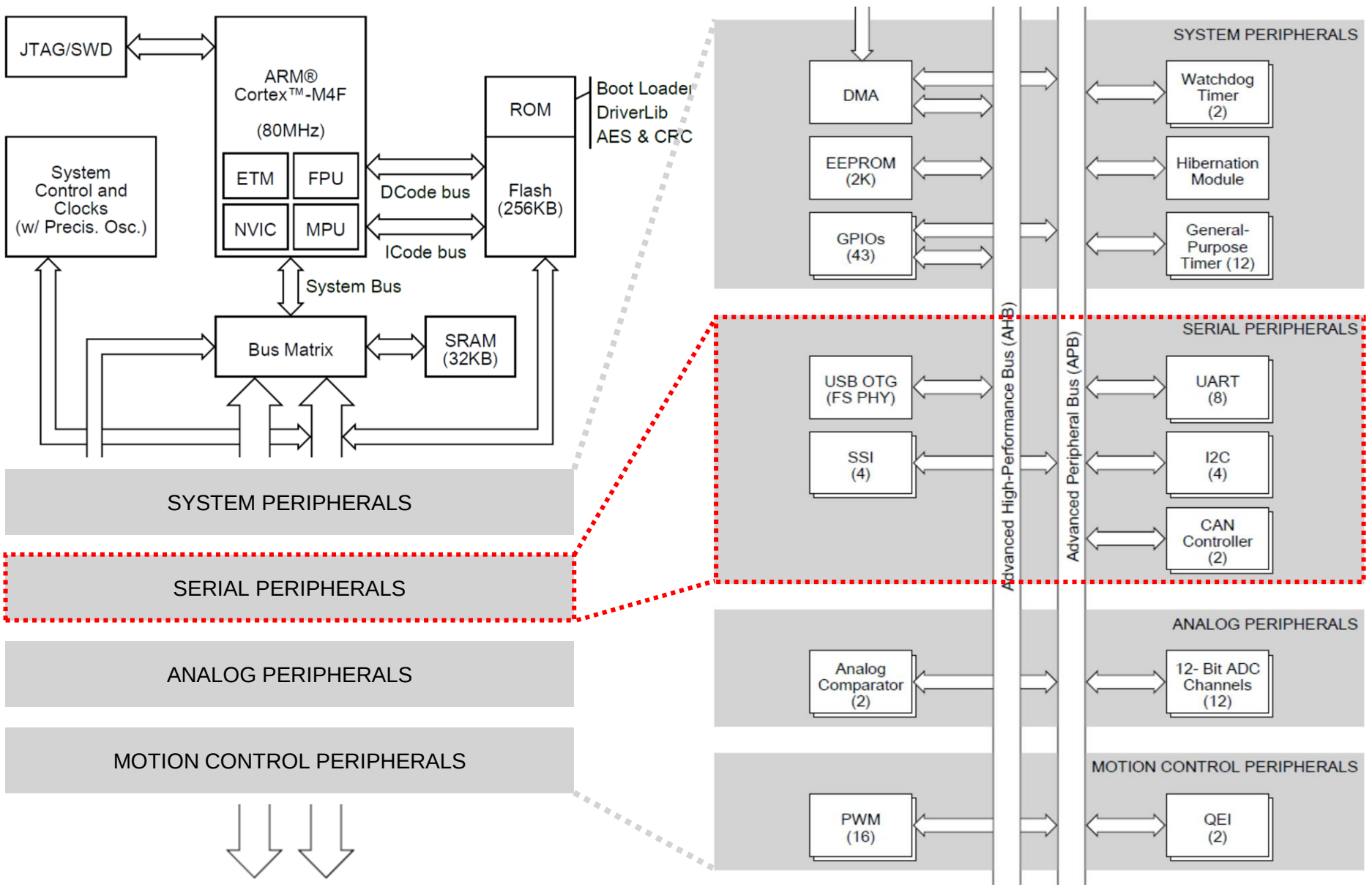
Lecture 06: Serial Communication (I)

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Thanks to Prof. Q. Xu and Drs. K. H. Wong, Philip Leong, Y.S. Moon, O. Mencer, N. Dulay, P. Cheung for some of the slides used in this course!

Recall: Tiva™ TM4C123GH6PM (MCU)

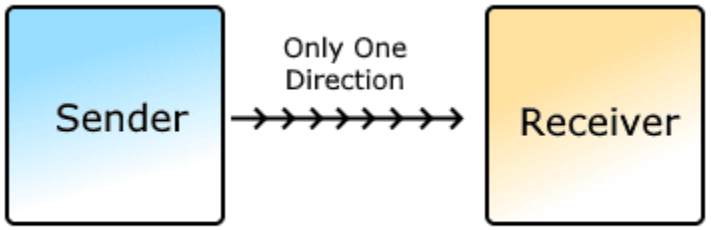


Outline

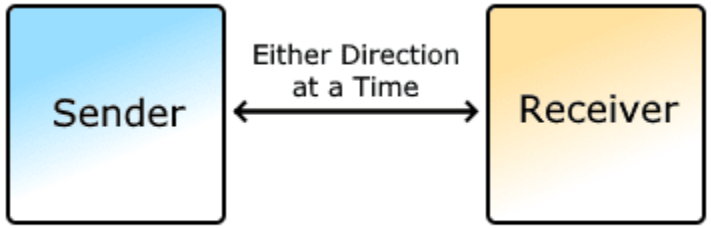
- **Basics**
  - Simplex vs. Half Duplex vs. Full Duplex
  - Parallel Communication
  - Serial Communication
    - Synchronous vs. Asynchronous
- **UART (Universal Asynchronous Receiver-Transmitter)**
  - Frame Format
  - Transmitter/Receiver Logic
- **Software Structures for Communication**
- **UART on Tiva™ & in TivaWare™ Library**
- **Preview: Lab05**

Communication

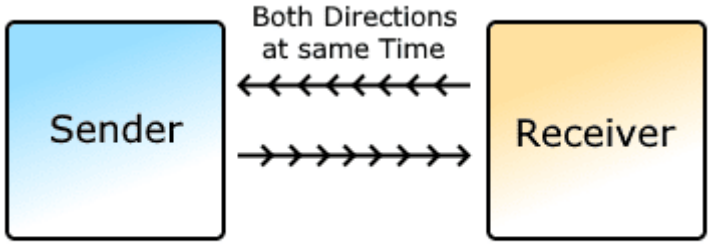
- Communication is the process of exchanging information between two parties.
- Depending on the modes of transmission, communication can be classified into:
  - ① **Simplex Communication**:
    - One-way, one direction, single channel.
  - ② **Half Duplex Communication**:
    - Two-way, one direction, single channel.
  - ③ **Full Duplex Communication**:
    - Two-way, both directions, two individual channels.



Simplex Communication



Half Duplex Communication



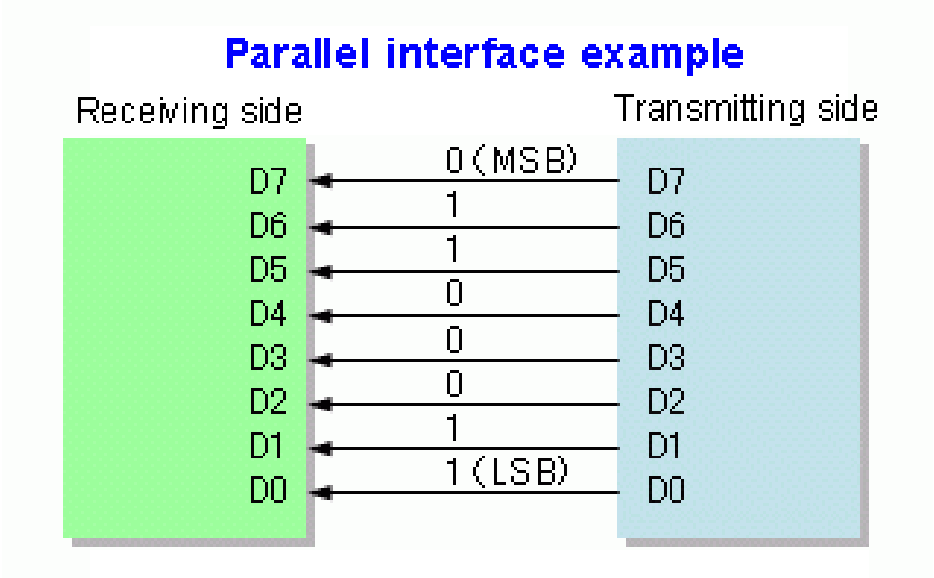
Full Duplex Communication

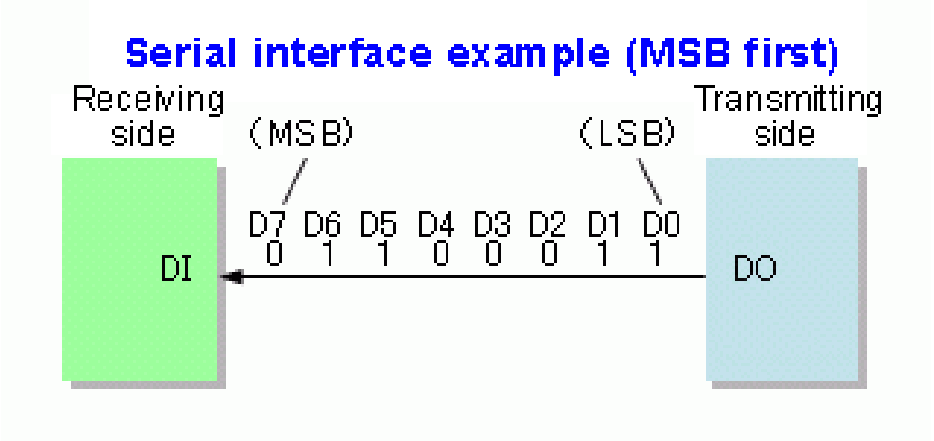
Parallel vs. Serial Communication

- Depending on the number of bits transferred at a time, communication can be further categorized into:

① **Parallel Communication:** conveys multiple bits of data simultaneously over parallel wires.

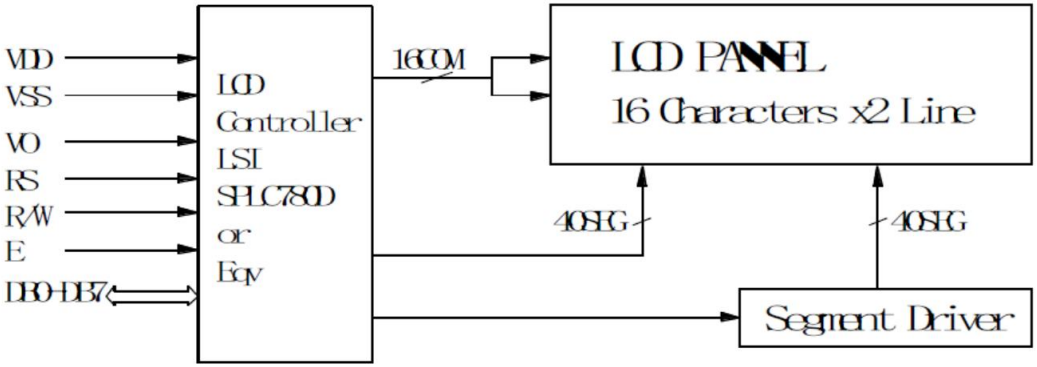
② **Serial Communication:** conveys data bit by bit over a single wire.



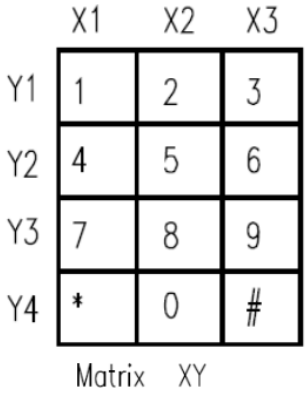


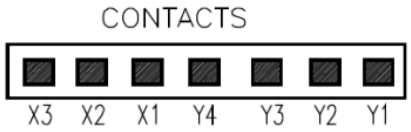
Recall: Lab04

- Tiva Launchpad interacts with the **Keypad** and **LCD display** via **???** **Communication**.



(3+8) pins are used for communication.



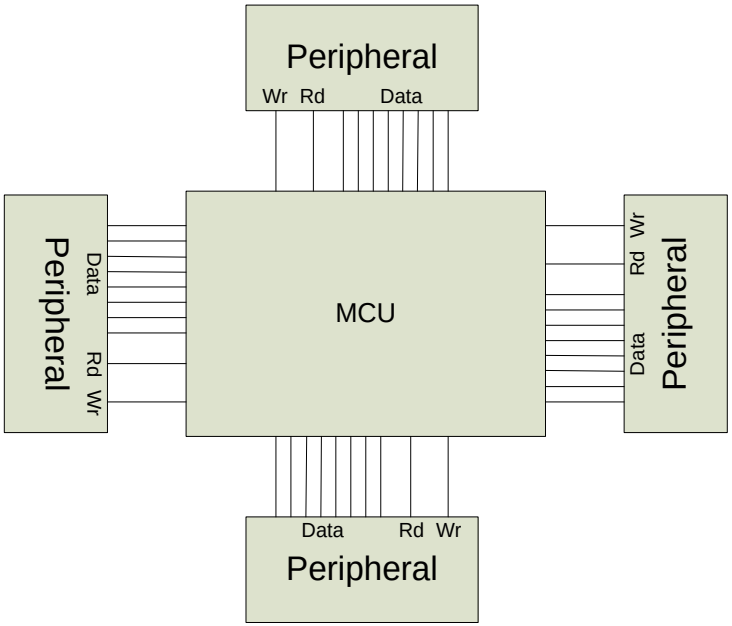


7 pins are used for communication.

Parallel: MCU-Peripheral Connections

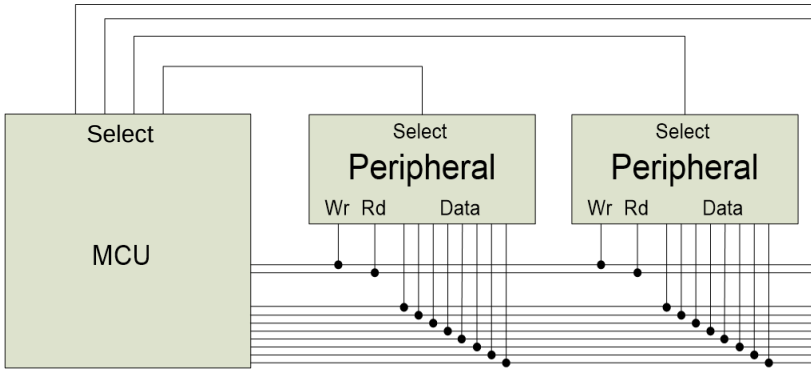
① Point-to-Point Connections:

- Direction connections between MCU and each peripheral;
- Simultaneous communications with multiple peripherals.



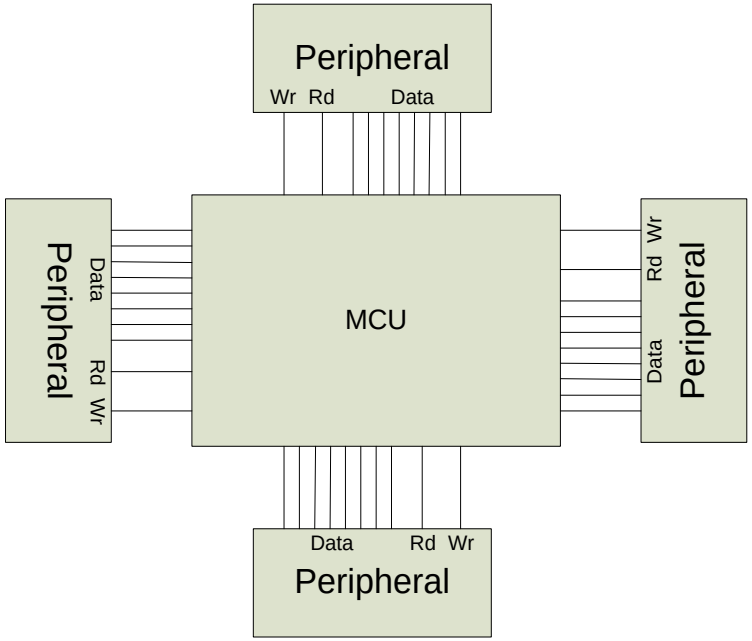
② Shared Buses Connections:

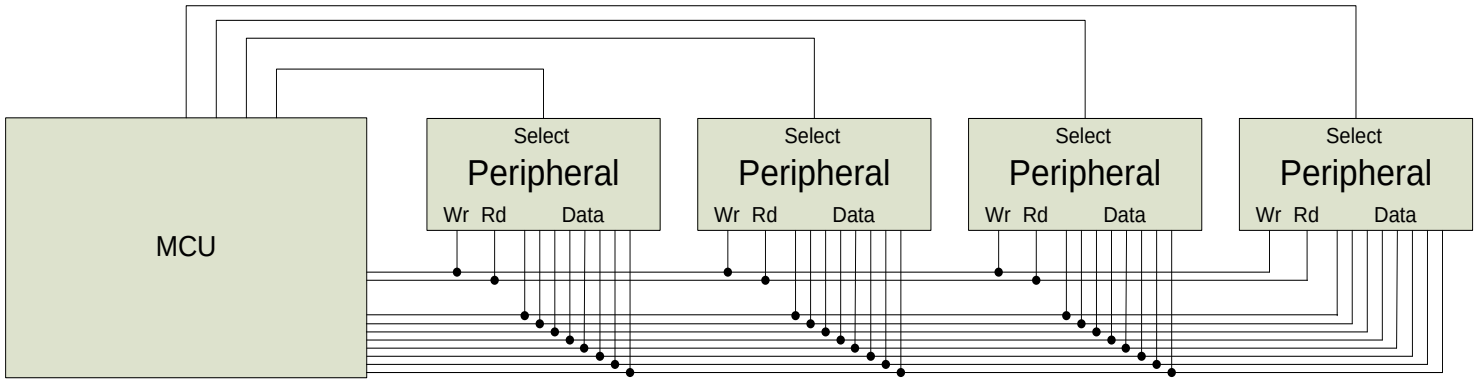
- **`Wr`/`Rd`/`Data`** lines shared by all peripherals;
- Individual **`Select`** line to address each peripheral;
- Communication with only one peripheral at a time.



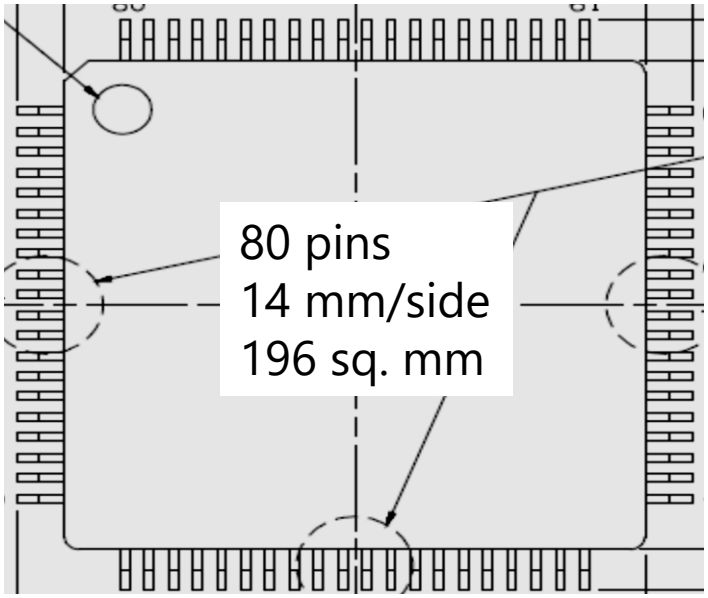
Class Exercise 6.1

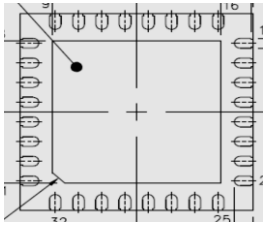
- Consider the **Point-to-Point** *(right)* and **Shared Buses Connections** *(below)*.
- Suppose the number of **`Data`** lines are 8. How many pins are needed to interconnect MCU with four peripherals?



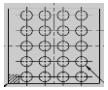


Why Communicate Serially?





32 pins
5 mm/side
25 sq. mm

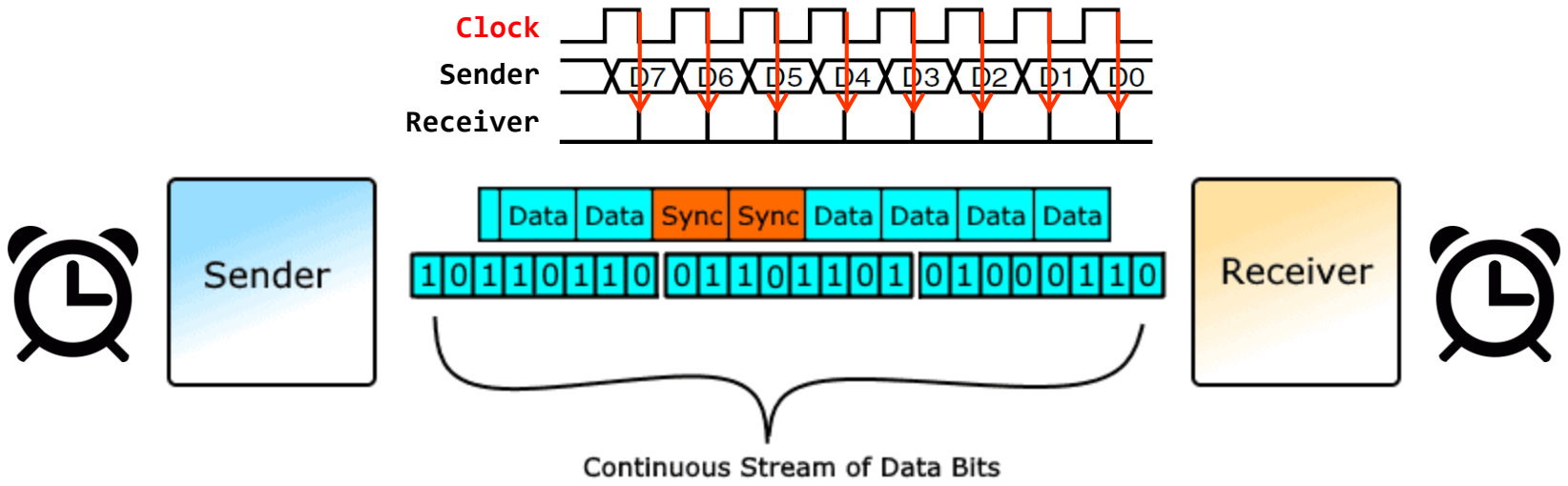


20 pins
1.94 mm/side
3.76 sq. mm

Pin count matters in embedded system!

Serial Communication (1/2)

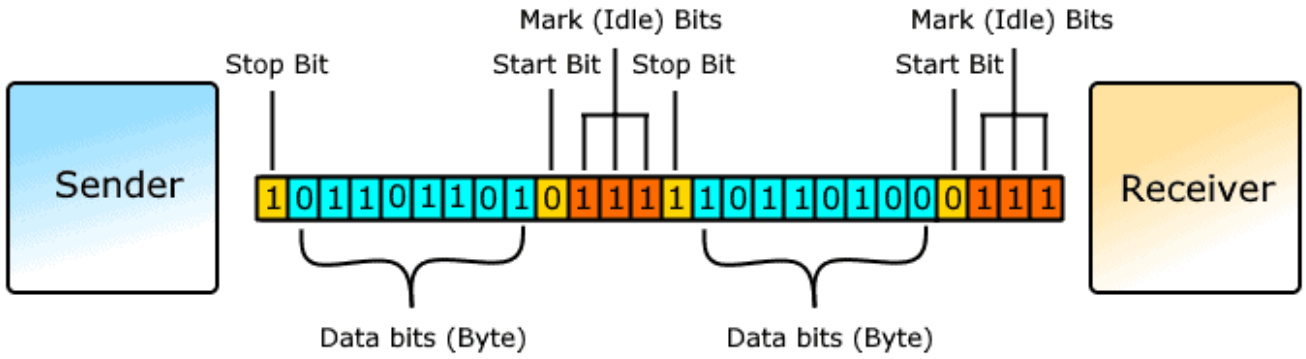
- **Serial comm.** can be **synchronous** or asynchronous:
  - **Synchronous Transmission**: The sender and receiver are synchronized using an external synchronization clock.



  - An extra channel is typically employed to transmit the **clock signal**.
    - It provides **synchronous clock pulses** that define a **constant time interval** for data transmission.
  - Common **synchronous** serial communication protocols: **Serial Peripheral Interface (SPI)** and **Inter-Integrated Circuit Bus (I^2C)**.

Serial Communication (2/2)

- Serial comm. can be **synchronous** or asynchronous:
  - **Asynchronous Transmission**: Additional bits are introduced to identify the beginning and end of the data.



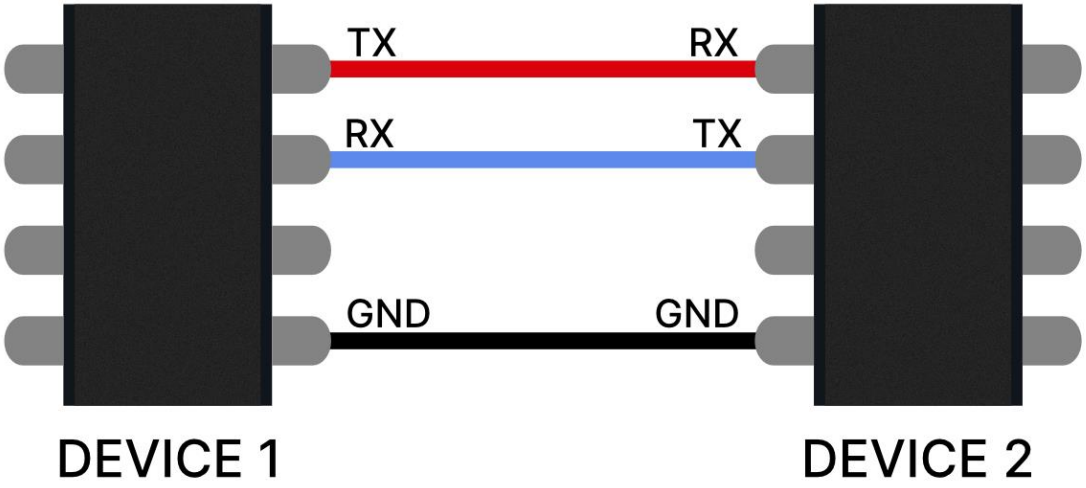
  - As a result, it does not need any external synchronization clock, but the sender and receiver must follow the same baud rate.
    - **Baud rate**: the number of symbols transmitted per second.
    - Higher baud rates allow faster transmission but are more error-prone.
  - Common **asynchronous** serial communication protocols: **Universal Asynchronous Receiver-Transmitter (UART)**, **Universal Serial Bus (USB)**, and **Bluetooth** (wireless!).

Outline

- Basics
  - Simplex vs. Half Duplex vs. Full Duplex
  - Parallel Communication
  - Serial Communication
    - Synchronous vs. Asynchronous
- **UART (Universal Asynchronous Receiver-Transmitter)**
  - Frame Format
  - Transmitter/Receiver Logic
- Software Structures for Communication
- UART on Tiva™ & in TivaWare™ Library
- Preview: Lab05

UART (Universal Asynchronous Receiver-Transmitter)

- It both **sends** & **receives** data (as its name suggests).
- It requires only two wires to communicate:
  - The transmitter (**TX**) wire on device 1 is connected to the receiver (**RX**) wire of device 2.
  - Likewise, the **TX** of device 2 connects to the **RX** of device 1.
  - Ground (**GND**) wire is needed to keep both devices at the same reference voltage.



How UART works (1/2)

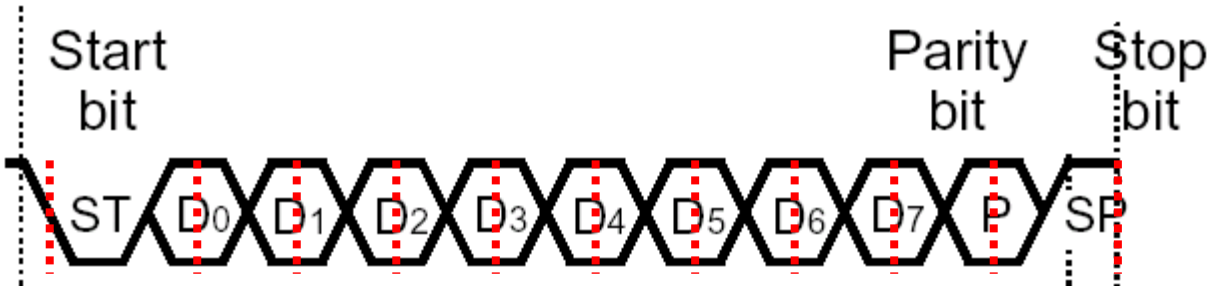
- UART communication is done in the so-called frames.
  - Each frame is made of bits including one start bit, 5 to 8 data bits, an *optional* parity bit, and one or two stop bits.

- **Transmitter (TX)**
  - If no data to send: keep sending 1.
    - *Why 1? This idea was taken from telegraphy, where the line held high indicated the transmitter was not damaged!*
  - When there is a data word to send:
    1. **Send** a `start bit` (`0`) to indicate the start of a word;
    2. **Send** `data bits` in the word, **LSB** first;
    3. **Send** `stop bit(s)` (‘1’s) to indicate the end of a word.

How UART works (2/2)

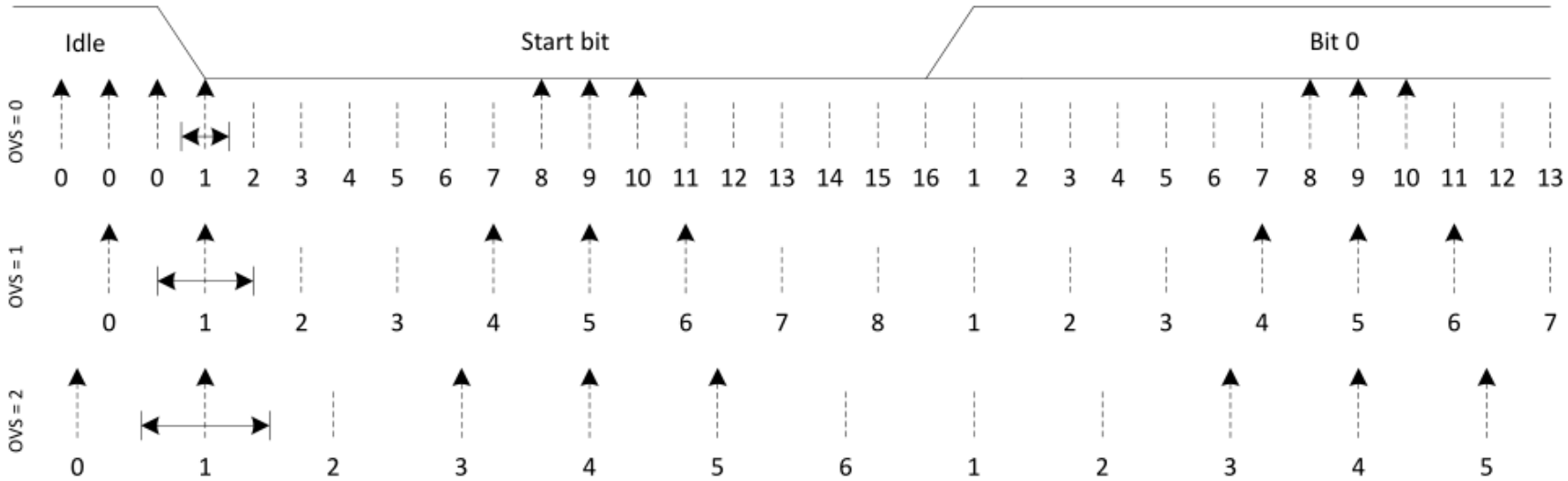
- **Receiver (RX)**
  - ① **Wait** for a falling edge (i.e., the beginning of the `start bit`) and wait for ½ bit time;
  - ② **Receive** the data (repeat as many `data bits` in a word):
    - **Wait** 1 bit time;
    - **Read** the data bit and shift it into a receiver buffer.
  - ③ **Wait** 1 bit time and **check** the bit value.
    - If 1 (i.e., the `stop bit`), then OK; otherwise, there's a problem!

Data Sampling Time at Receiver



UART: Data Oversampling

- At the receiver (RX) side, UART usually *oversamples* the incoming data line.
  - Extra samples allow majority voting, improving noise immunity.
    - Each bit is sampled three times at the middle of the bit period.
    - If two of the three samples are high, the bit will be recorded as high.



For OVS = 0 (16x oversampling), these samples are taken at positions 8, 9, and 10.
For OVS = 1 (8x oversampling), samples are taken at 4, 5, and 6.
For OVS = 2 (6x oversampling), samples are taken at 3, 4, and 5.
For OVS = 3 (4x oversampling), majority voting is not used.

UART: (Optional) Parity Bit

- The UART `parity bit` can be defined as:
  - **`None`**: meaning there is no parity bit;
  - **`Stick`**: setting to `0` or `1` if the total number of logic '1's in the data is an even or odd number, respectively.
  - **`Even`** or **`Odd`**: an extra bit is stuffed to make the total number of logic '1's in the data plus the parity bit even or odd.
  - E.g., there are 6 '1's in "`01110111`", so the parity bit will be '`0`' for **`even`** parity, '`1`' for **`odd`** parity, and '`0`' for **`stick`** parity.

Class Exercise 6.2

- Suppose that we are transmitting the character ‘D’ (binary `01000100`) using UART with 8 data bits, odd parity bit, and 2 stop bits, show the sequence of bits transferred.

Outline

- **Basics**
  - Simplex vs. Half Duplex vs. Full Duplex
  - Parallel Communication
  - Serial Communication
    - Synchronous vs. Asynchronous
- **UART (Universal Asynchronous Receiver-Transmitter)**
  - Frame Format
  - Transmitter/Receiver Logic
- **Software Structures for Communication**
- **UART on Tiva™ & in TivaWare™ Library**
- **Preview: Lab05**

Polling vs. Interrupt

- **Polling**
  - Spin in a loop until it is ready to send/receive data.
  - Hard to share the µP.

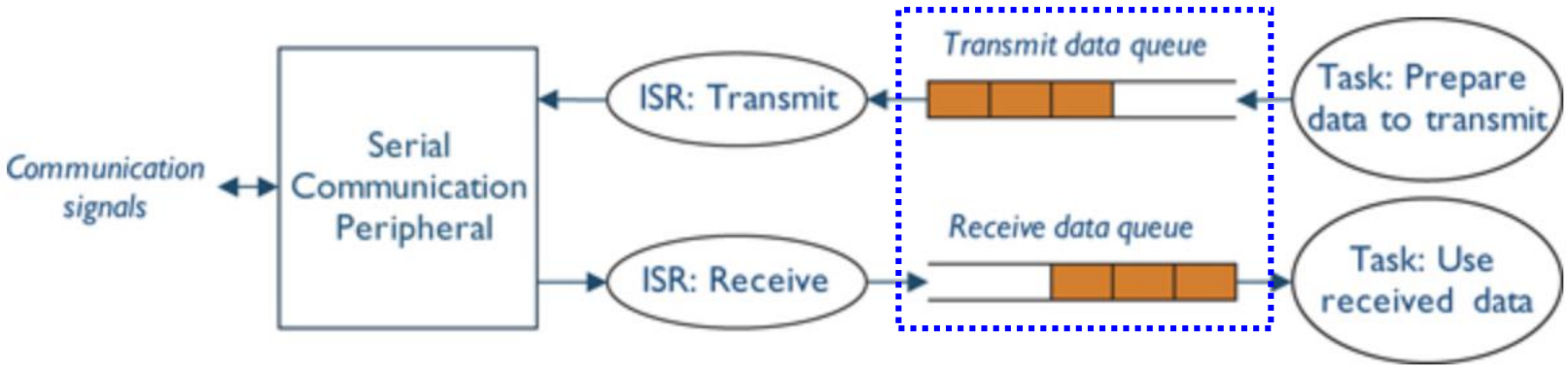
```
Main Program:
  while (TRUE)
  {
    if ready
      Send/Receive data;
    if time elapsed
      Do some task;
  }
```

- **Interrupt**
  - Signal an interrupt whenever it is ready.
  - µP runs the handler to send/receive data.

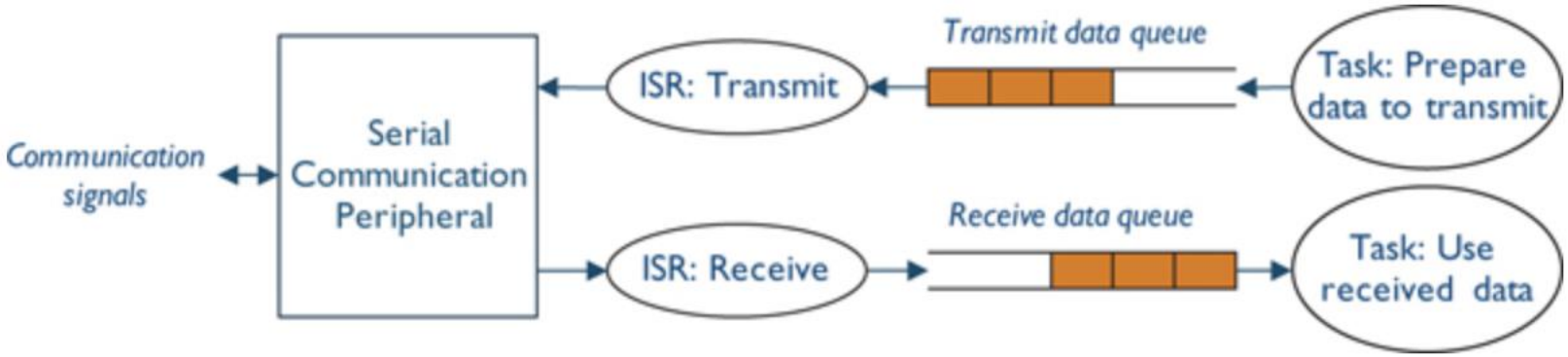
```
Main Program:
  Do some task;

Interrupt Handler:
  Clear the interrupt;
  Send/Receive data;
```

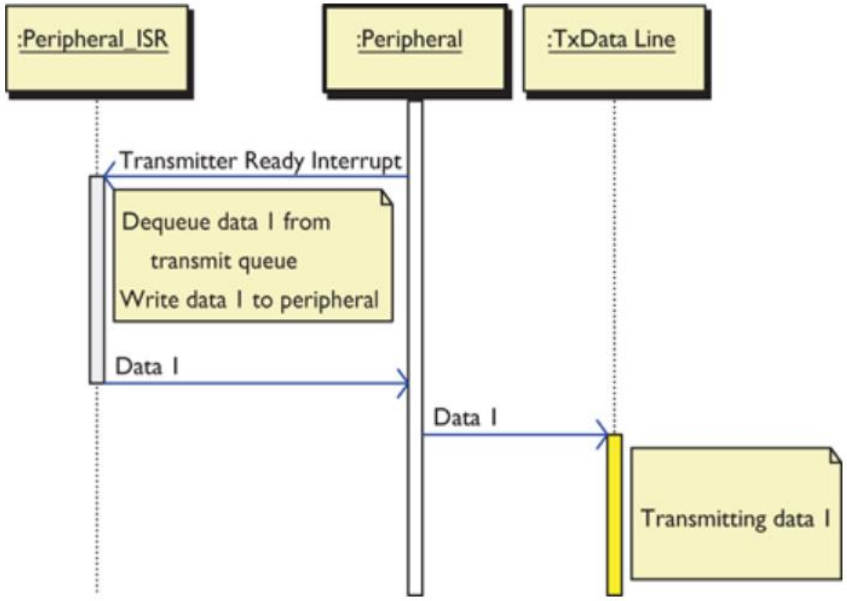
Interrupt-Driven Queued Comm. (1/2)

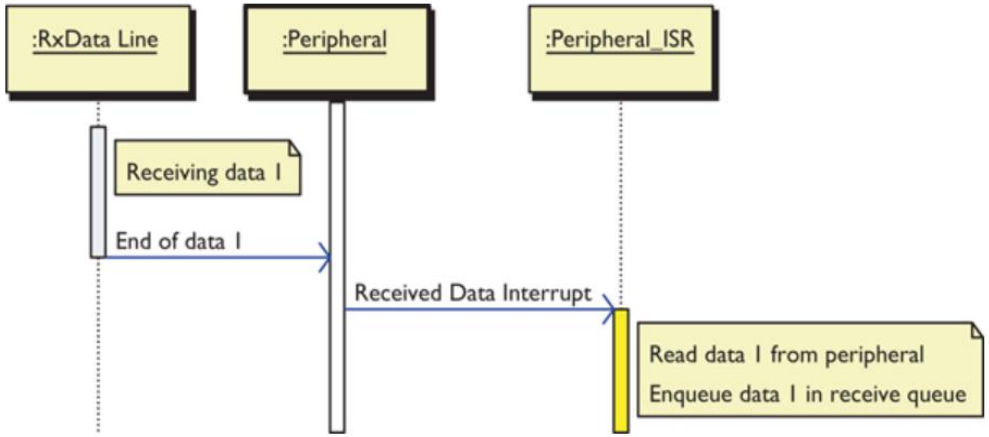
- Recall that the longer an ISR takes to run, the longer all other ISRs can be delayed.
  - The ISR must perform the most time-critical operations with the data and leave other processing for lower-priority code.
- To this end, we must somehow keep data temporarily.
  - In most cases, data should be delivered in the order it was transmitted, following the first-in, first-out (FIFO) ordering.
  - A queue is another name for a buffer with FIFO ordering.



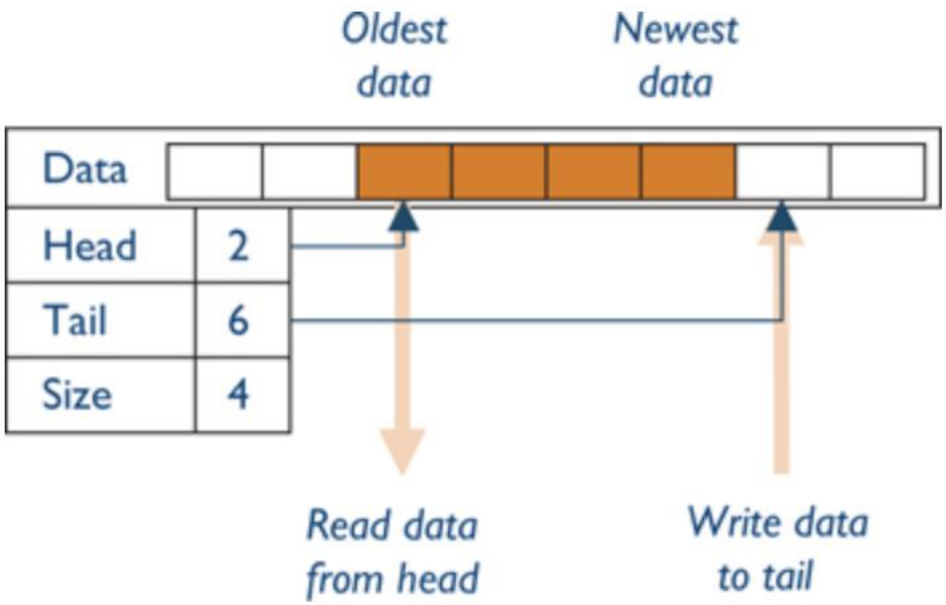
Interrupt-Driven Queued Comm. (2/2)







Queue Implementation



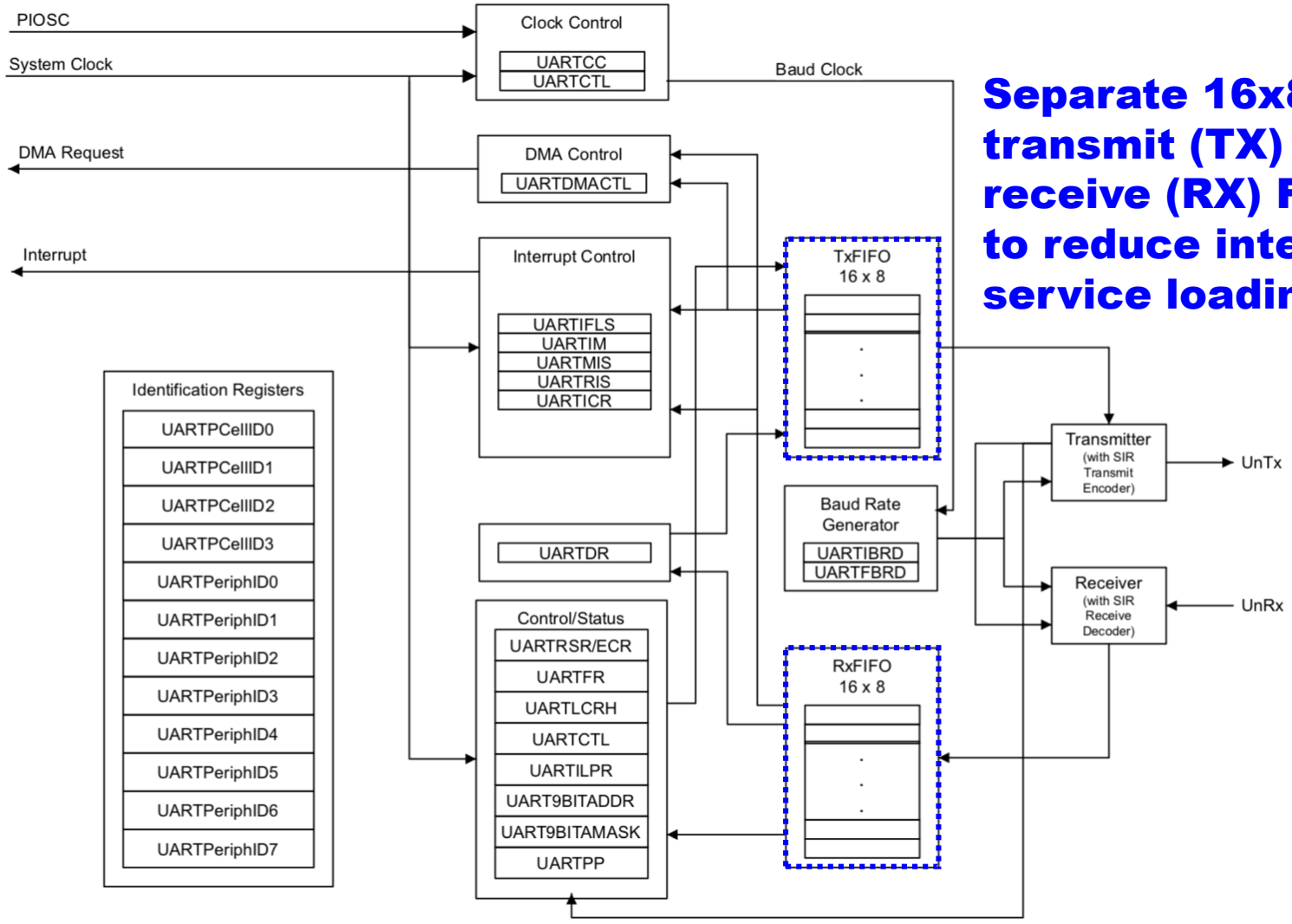
- We can implement a queue using an array.
  - ~~Rather than move all data~~ each time an item is added or removed, we use indexes to keep track of the head and tail.
    - **Head**: Indicate the oldest data item (to be **dequeued**).
    - **Tail**: Indicate the free space to use when **enqueueing** the next item.
    - **Size**: Indicate the number of items in use.

Outline

- **Basics**
  - Simplex vs. Half Duplex vs. Full Duplex
  - Parallel Communication
  - Serial Communication
    - Synchronous vs. Asynchronous
- **UART (Universal Asynchronous Receiver-Transmitter)**
  - Frame Format
  - Transmitter/Receiver Logic
- **Software Structures for Communication**
- **UART on Tiva™ & in TivaWare™ Library**
- **Preview: Lab05**

UART Module on Tiva™

- There are eight UART modules on Tiva:



Separate 16x8 transmit (TX) and receive (RX) FIFOs to reduce interrupt service loading!

UART Signals on Tiva™

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

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

UART in TivaWare™ Library (1/2)

30 UART

30.1 Introduction

The Universal Asynchronous Receiver/Transmitter (UART) API provides a set of functions for using the Tiva UART modules. Functions are provided to configure and control the UART modules, to send and receive data, and to manage interrupts for the UART modules.

The Tiva UART performs the functions of parallel-to-serial and serial-to-parallel conversions. It is very similar in functionality to a 16C550 UART, but is not register-compatible.

Some of the features of the Tiva UART are:

- A 16x12 bit receive FIFO and a 16x8 bit transmit FIFO.
- Programmable baud rate generator.
- Automatic generation and stripping of start, stop, and parity bits.
- Line break generation and detection.
- Programmable serial interface
  - 5, 6, 7, or 8 data bits
  - even, odd, stick, or no parity bit generation and detection
  - 1 or 2 stop bit generation
  - baud rate generation, from DC to processor clock/16
- Modem control/flow control
- IrDA serial-IR (SIR) encoder/decoder.
- uDMA interface
- 9-bit operation

This driver is contained in `driverlib/uart.c`, with `driverlib/uart.h` containing the API declarations for use by applications.

30.2 API Functions

Functions

- void UART9BitAddrSend (uint32_t ui32Base, uint8_t ui8Addr)
- void UART9BitAddrSet (uint32_t ui32Base, uint8_t ui8Addr, uint8_t ui8Mask)
- void UART9BitDisable (uint32_t ui32Base)
- void UART9BitEnable (uint32_t ui32Base)
- void UARTBreakCtl (uint32_t ui32Base, bool bBreakState)
- bool UARTBusy (uint32_t ui32Base)

- int32_t UARTCharGet (uint32_t ui32Base)
- int32_t UARTCharGetNonBlocking (uint32_t ui32Base)
- void UARTCharPut (uint32_t ui32Base, unsigned char ucData)
- bool UARTCharPutNonBlocking (uint32_t ui32Base, unsigned char ucData)
- bool UARTCharsAvail (uint32_t ui32Base)
- uint32_t UARTClockSourceGet (uint32_t ui32Base)
- void UARTClockSourceSet (uint32_t ui32Base, uint32_t ui32Source)
- void UARTConfigGetExpClk (uint32_t ui32Base, uint32_t ui32UARTClk, uint32_t *pui32Baud, uint32_t *pui32Config)
- void UARTConfigSetExpClk (uint32_t ui32Base, uint32_t ui32UARTClk, uint32_t ui32Baud, uint32_t ui32Config)
- void UARTDisable (uint32_t ui32Base)
- void UARTDisableSIR (uint32_t ui32Base)
- void UARTDMADisable (uint32_t ui32Base, uint32_t ui32DMAFlags)
- void UARTDMAEnable (uint32_t ui32Base, uint32_t ui32DMAFlags)
- void UARTEnable (uint32_t ui32Base)
- void UARTEnableSIR (uint32_t ui32Base, bool bLowPower)
- void UARTFIFODisable (uint32_t ui32Base)
- void UARTFIFOEnable (uint32_t ui32Base)
- void UARTFIFOLevelGet (uint32_t ui32Base, uint32_t *pui32TxLevel, uint32_t *pui32RxLevel)
- void UARTFIFOLevelSet (uint32_t ui32Base, uint32_t ui32TxLevel, uint32_t ui32RxLevel)
- uint32_t UARTFlowControlGet (uint32_t ui32Base)
- void UARTFlowControlSet (uint32_t ui32Base, uint32_t ui32Mode)
- void UARTIntClear (uint32_t ui32Base, uint32_t ui32IntFlags)
- void UARTIntDisable (uint32_t ui32Base, uint32_t ui32IntFlags)
- void UARTIntEnable (uint32_t ui32Base, uint32_t ui32IntFlags)
- void UARTIntRegister (uint32_t ui32Base, void (*pfnHandler)(void))
- uint32_t UARTIntStatus (uint32_t ui32Base, bool bMasked)
- void UARTIntUnregister (uint32_t ui32Base)
- void UARTLoopbackEnable (uint32_t ui32Base)
- void UARTModemControlClear (uint32_t ui32Base, uint32_t ui32Control)
- uint32_t UARTModemControlGet (uint32_t ui32Base)
- void UARTModemControlSet (uint32_t ui32Base, uint32_t ui32Control)
- uint32_t UARTModemStatusGet (uint32_t ui32Base)
- uint32_t UARTParityModeGet (uint32_t ui32Base)
- void UARTParityModeSet (uint32_t ui32Base, uint32_t ui32Parity)
- void UARTRxErrorClear (uint32_t ui32Base)
- uint32_t UARTRxErrorGet (uint32_t ui32Base)
- void UARTSmartCardDisable (uint32_t ui32Base)
- void UARTSmartCardEnable (uint32_t ui32Base)
- bool UARTSpaceAvail (uint32_t ui32Base)
- uint32_t UARTTxIntModeGet (uint32_t ui32Base)
- void UARTTxIntModeSet (uint32_t ui32Base, uint32_t ui32Mode)

UART in TivaWare™ Library (2/2)

- `UARTConfigSetExpClk`: sets the UART configuration.
  - E.g., the base address, the rate of the clock supplied, the desired baud rate, the frame format.

Transmitter (TX)

- `UARTSpaceAvail`: checks if there is any space in the **TX FIFO**.
- `UARTCharPut`: sends the character to the **TX FIFO**.

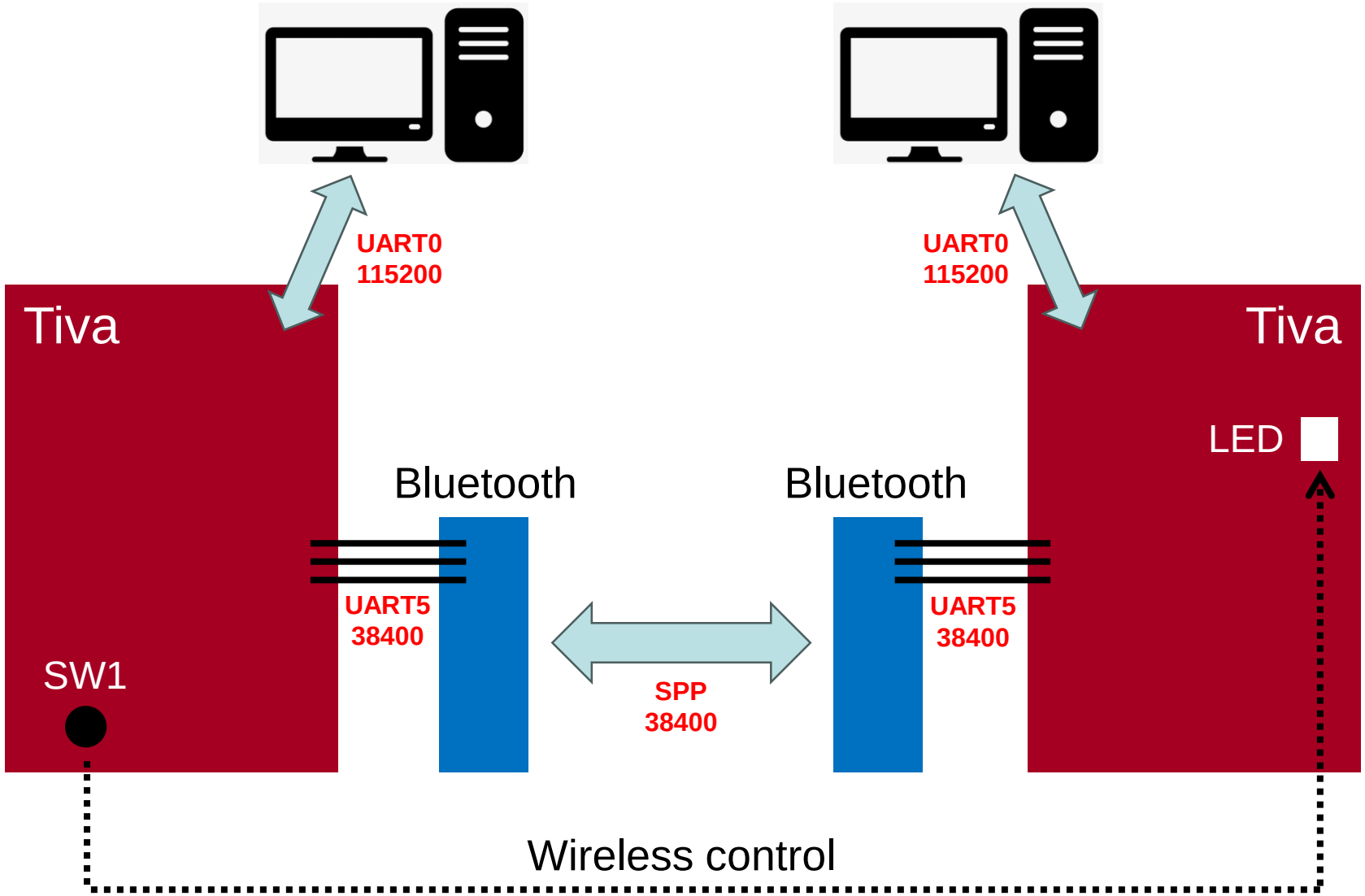
Receiver (RX)

- `UARTCharsAvail`: checks if there are any char. in the **RX FIFO**.
- `UARTCharGet`: gets a character from the **RX FIFO**.

Outline

- **Basics**
  - Simplex vs. Half Duplex vs. Full Duplex
  - Parallel Communication
  - Serial Communication
    - Synchronous vs. Asynchronous
- **UART (Universal Asynchronous Receiver-Transmitter)**
  - Frame Format
  - Transmitter/Receiver Logic
- **Software Structures for Communication**
- **UART on Tiva™ & in TivaWare™ Library**
- **Preview: Lab05**

Preview: Lab05



Summary

- **Basics**
  - Simplex vs. Half Duplex vs. Full Duplex
  - Parallel Communication
  - Serial Communication
    - Synchronous vs. Asynchronous
- **UART (Universal Asynchronous Receiver-Transmitter)**
  - Frame Format
  - Transmitter/Receiver Logic
- **Software Structures for Communication**
- **UART on Tiva™ & in TivaWare™ Library**
- **Preview: Lab05**