مدينة زويل للعلوم والتكنولوجيا
Zewail City of Science and Technology

COMMUNICATION AND INFORMATION ENGINEERING

CIE 314
Embedded Systems Fundamentals

Lecture #6
Interrupts and Program Design

Instructor:

Dr. Ahmad El-Banna

SPRING 2017

Agenda

Introduction to Interrupts

- Device Services: Polling Versus Interrupts
- Interrupt Servicing Events
- What is Reset?

Notes on Program Design

Design Tips

3.9 INTRODUCTION TO INTERRUPTS

- Device Services: Polling Versus Interrupts
- Interrupt Servicing Events
- What is Reset?

POLLING VS. INTERRUPT SERVICING

- Service by Polling
  - The CPU periodically interrogates a device status to learn if it need service
  - Usually done in a loop
  - When need is identified, CPU executes service instructions
- Service by Interrupt
  - The device notifies the CPU when service is needed
    - Interrupt request
  - Meanwhile, the CPU can be used for other tasks or sent to sleep mode
  - When service request arrives, CPU interrupts what it is doing to execute the service instructions (Interrupt Service Routine – ISR)
  - Upon completion (IRET) control is transferred back to interrupted task

INTERRUPT TYPES

- Maskable Interrupt
  - Can be blocked by clearing the GIE flag
  - Most common type of interrupt
  - GIE cleared by RESET
  - GIE automatically cleared when entering an ISR
- Non-maskable Interrupt (NMI)
  - Cannot be blocked by clearing GIE
  - Reserved for system critical events
  - Requires an ISR to serve the critical event

INTERRUPT SERVICING EVENTS

- **Step 1:** Finish the instruction being executed
- **Step 2:** Save the current PC value and the status register (SR), onto the stack
- **Step 3:** Clear the global interrupt enable flag
- **Step 4:** Load the program counter (PC) with the address of the ISR to be executed
- **Step 5:** Execute the corresponding ISR
  - A vector number is used to identify the corresponding ISR
- **Step 6:** Restore the program counter and status that were saved onto the stack in Step 2
  - Control transferred back to interrupted task

INTERRUPT SUPPORT INFRASTRUCTURE

- A means for saving the program counter and status register
  - Provided through a properly allocated stack
- An interrupt service routine designed to render the requested service
  - Provided by a properly designed ISR
- A means for locating the ISR corresponding to a particular request
  - Provided by a properly initialized vector table
- An enabled interrupts structure
  - Achieved by setting the GIE and the peripheral device IF

WHAT IS A RESET?

- An asynchronous signal causing the CPU to start from a predefined known state
- Power-on Reset
  - Generated when power is applied for the first time
  - Loads PC with “Reset Address”
  - Clears Status register, including GIE
- Other Reset Triggers
  - Events that might compromise the system integrity
  - External hardware events (like a reset pushbutton depress)
  - Expiration of watchdog timer
  - Assertion of the CPU reset pin

3.10 NOTES ON PROGRAM DESIGN

PROGRAMMING RECOMMENDATIONS

- Program Planning & Design
  - Plan your program before start coding
- Document Your Design
  - Begin documenting since the planning phase
- Partition Your Design
  - Design modular programs
- Code in the Selected Language
  - Be aware of syntax rules and language features
- Integrate, Test, & Debug
  - Join modules one adding one at a time
  - Test under different conditions
  - Correct bugs as they appear

EMBEDDED SW RECOMMENDATIONS (1/2)

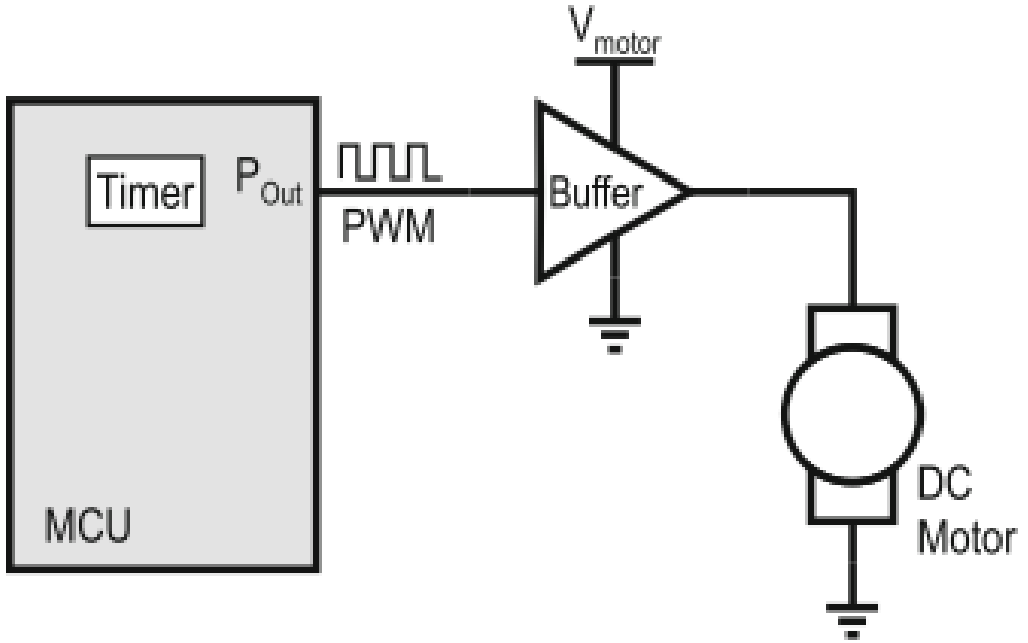
- Write the shortest possible program
  - Short programs are easier to understand and modify
- Evaluate shortcuts before implementing them
  - Avoid programming with tricks
- Write programs easy to understand
  - Document well to later understand the code
- Write programs easy to modify
  - Makes it easier maintaining the code
- Document your program as you work on it
  - Will make code easy to modify and reusable
  - Never leave the documentation for last

EMBEDDED SW RECOMMENDATIONS (2/2)

- Create your own catalog of functions for common tasks
  - Increases code reusability and productivity
- Program for lowest power consumption
  - Activate the low-power modes in your hardware
  - Be self conscious about code efficiency
- Know your hardware system
  - Embedded software needs to match your hardware
- Avoid using unimplemented bits of memory or registers
  - Reduces the risk of early code obsolescence
- Whenever possible, use a single resource for a single purpose
  - Minimizes chances of bugs due to confusion in usage

DESIGN TIPS

Fig. 8.58 Open-loop DC motor speed control via PWM



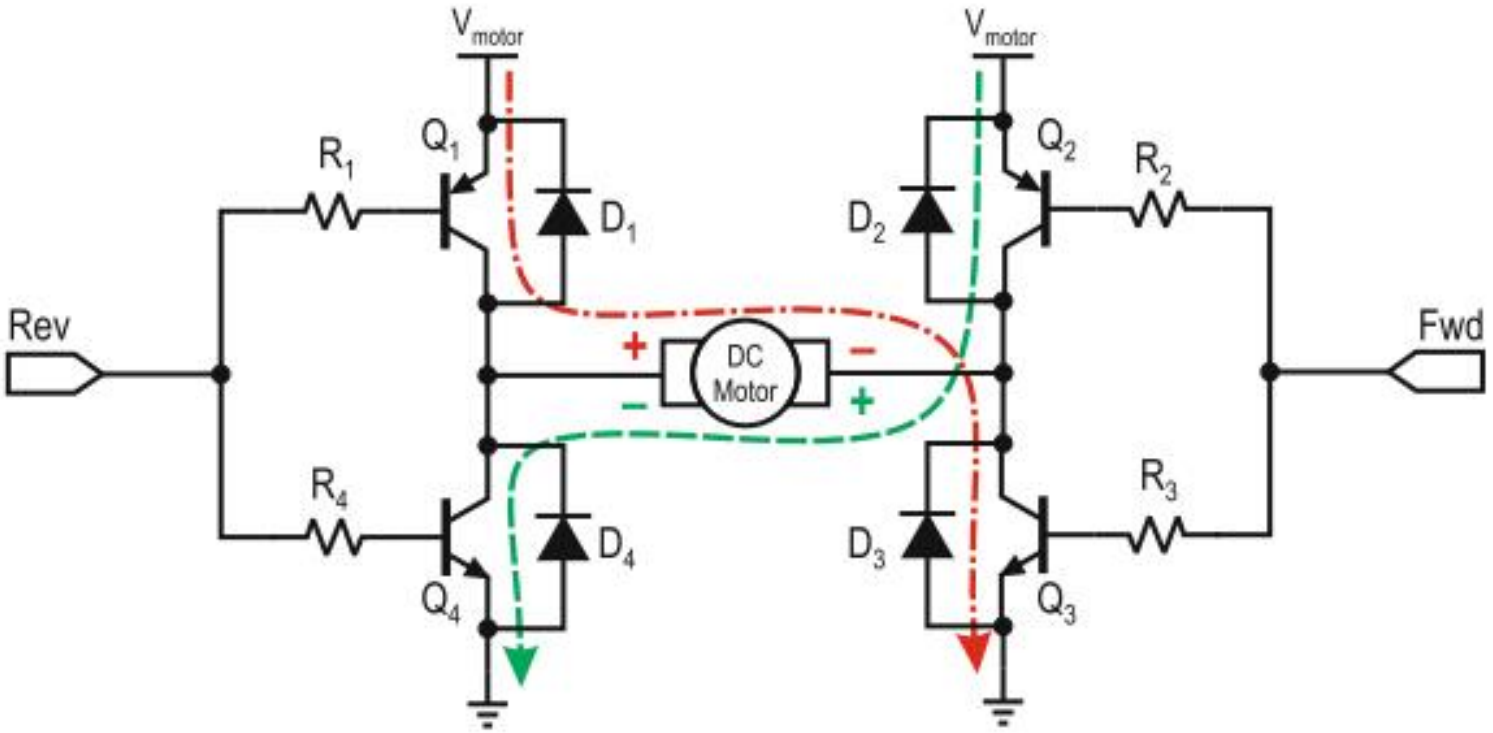
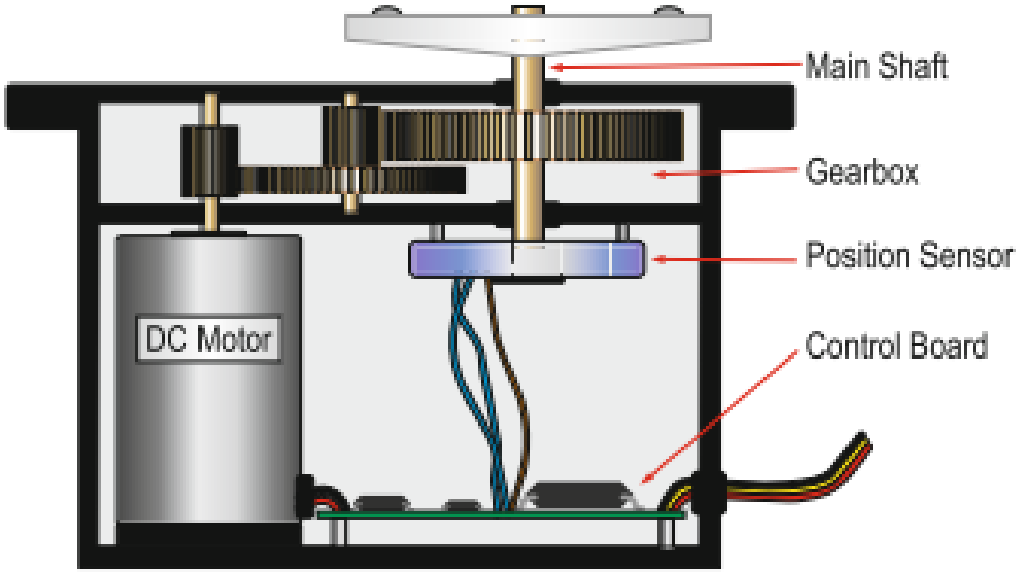


Fig. 8.59 Reversible speed H-bridge DC motor driver

Fig. 8.60 Pictorial view of internals of a small servomotor



- A **servo-motor** is a DC motor with feedback control that allows for precise position control.
- 3wires :
  - VDD-Gnd→ power
  - Control→ PWM signal (typical period 20 ms & Width of the ON specifies the desired angular position)
- Interfacing to an MCU requires a single PWM output, and the control pin in most cases can be directly driven from an I/O pin or simply a level shifter.

A stepper motor shaft moves in discrete increments in response to digital pulse sequences applied from a controller.

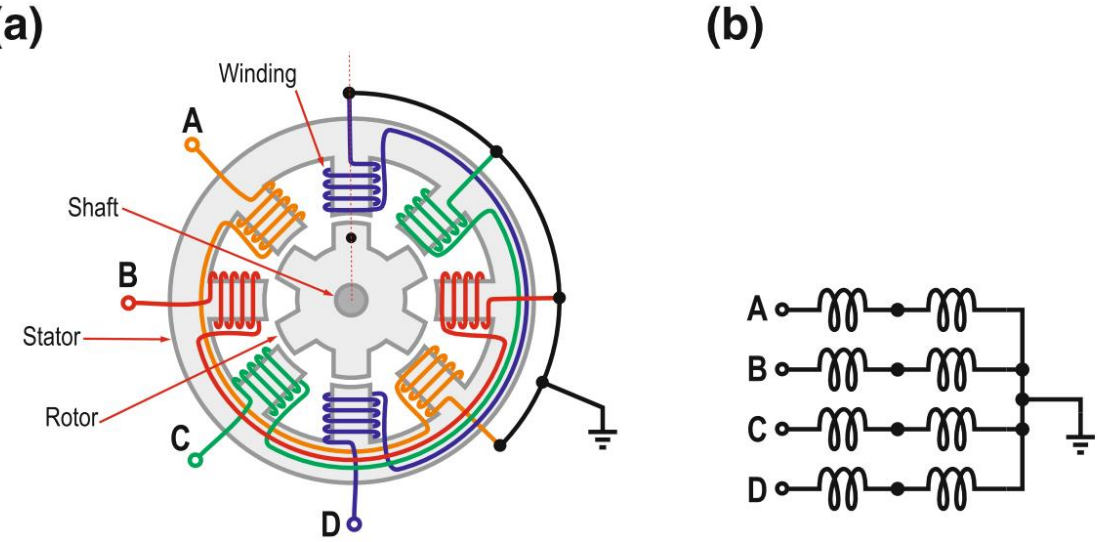


Fig. 8.61 Sectional view of a variable reluctance stepper motor and winding arrangement. **a** Motor cross section, **b** Winding arrangement

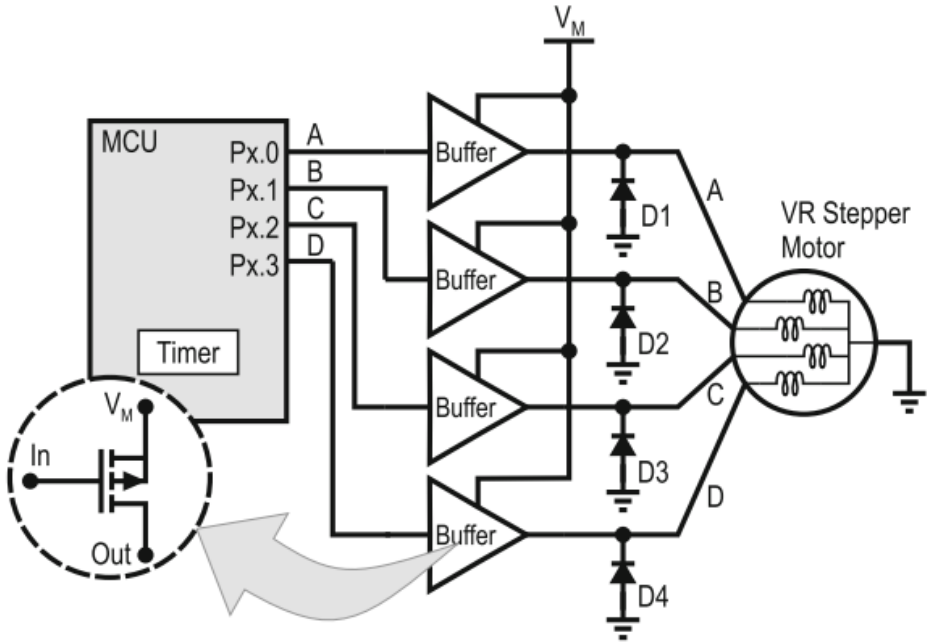


Fig. 8.64 Interface block diagram for a 4-phase VR stepper motor

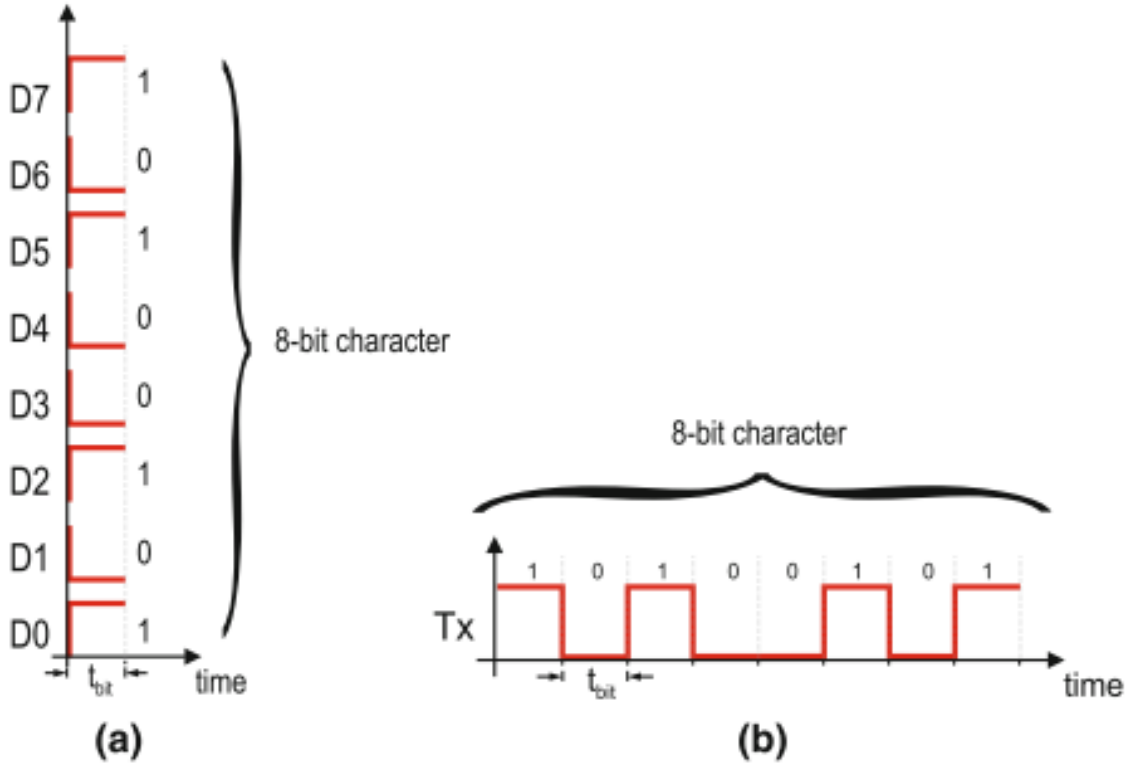


Fig. 9.1 Signal diagrams for serial and parallel communication channels. **a** Parallel. **b** Serial

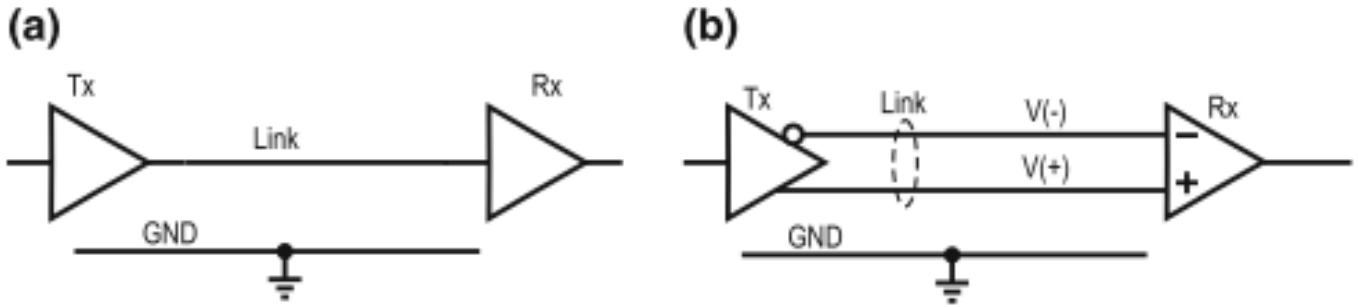


Fig. 9.2 Wired connection modalities for serial channels. **a** Single ended. **b** Differential

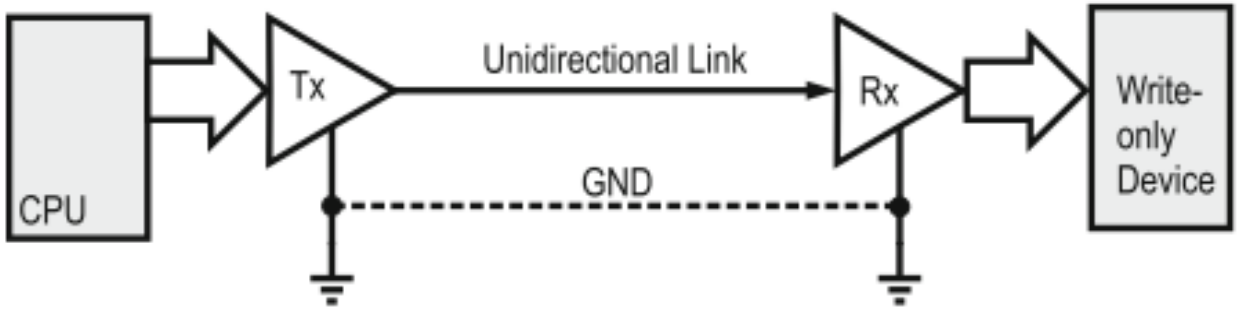


Fig. 9.3 Simplex serial channel connecting a CPU to a write-only device

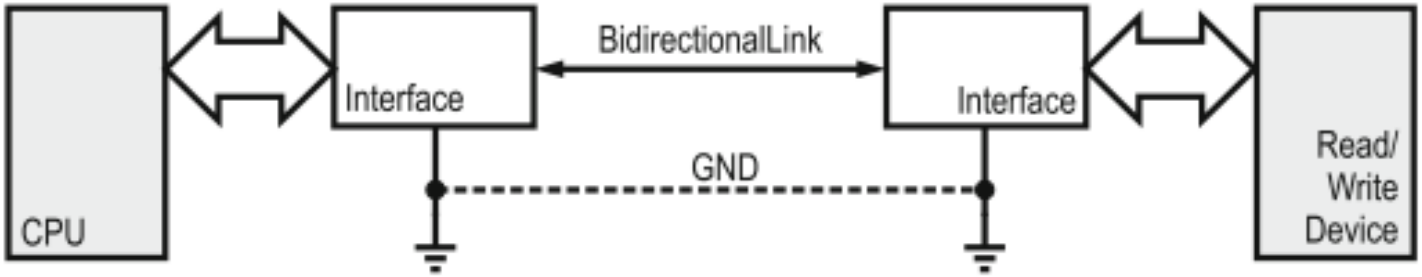


Fig. 9.4 Half-duplex serial channel connecting a CPU to a read/write device

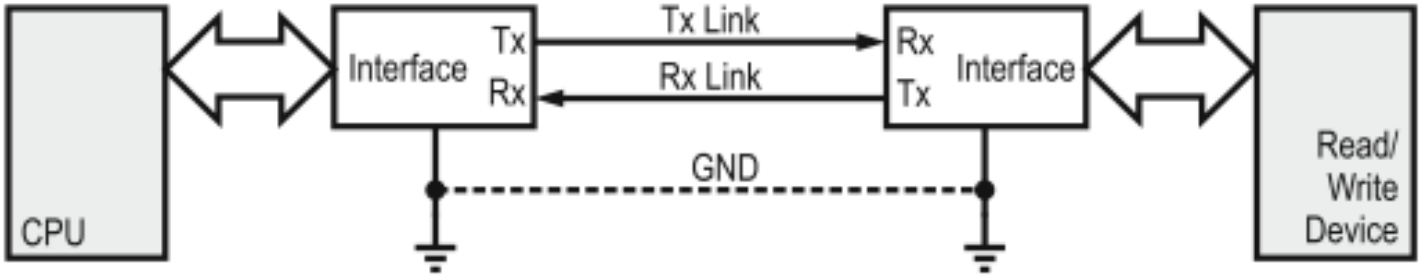


Fig. 9.5 A full-duplex serial channel connecting a CPU to a read/write peripheral

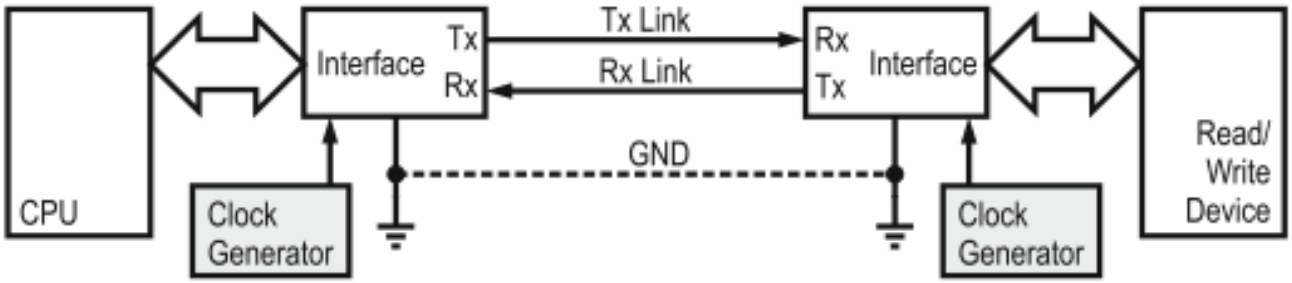


Fig. 9.6 An asynchronous serial channel denoting independent clock sources at channel ends

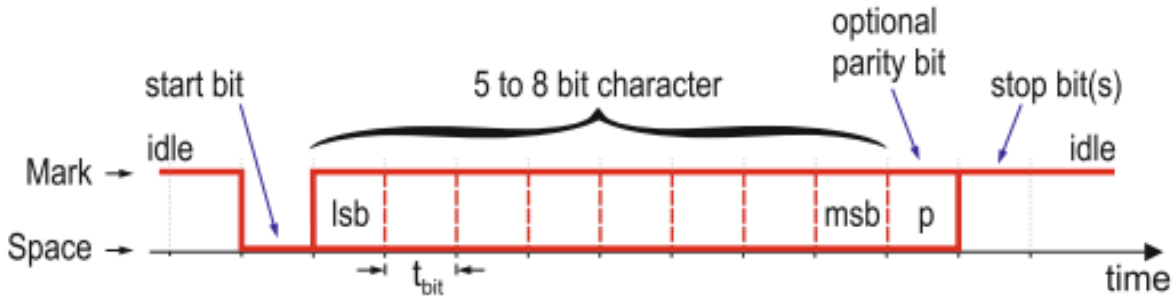


Fig. 9.7 Format of an asynchronous packet

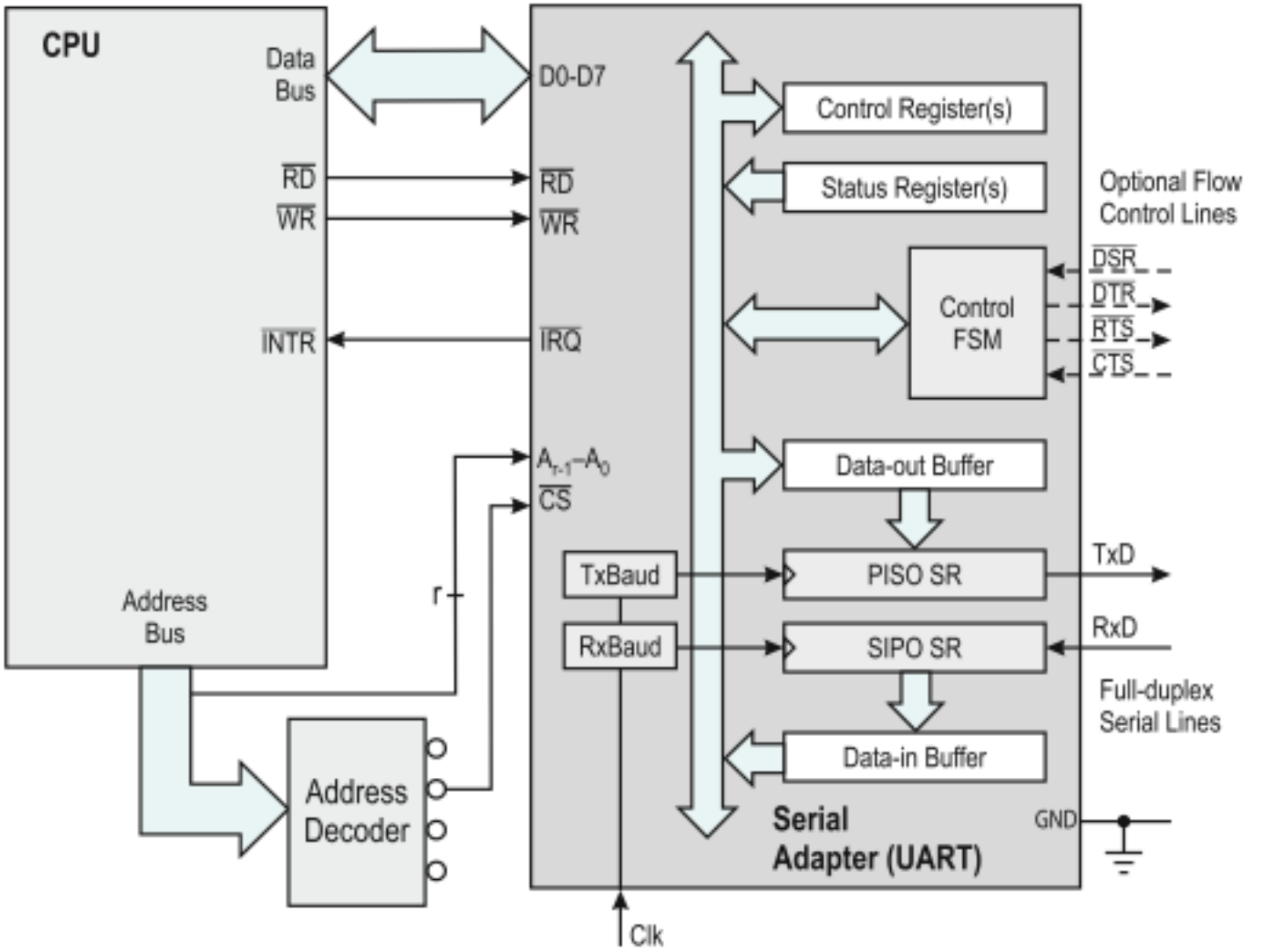


Fig. 9.9 General organization of a Universal Asynchronous Receiver and Transmitter (UART)

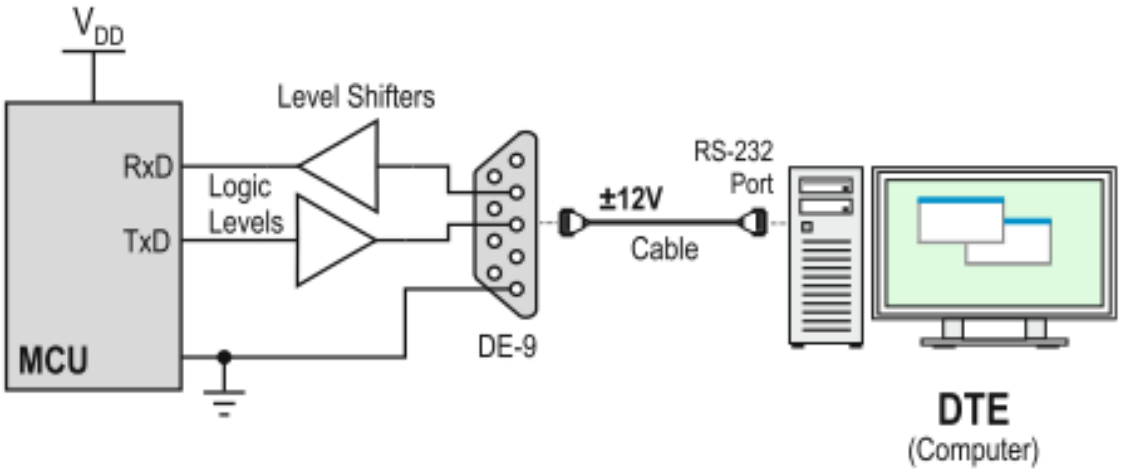


Fig. 9.10 Level shifting for physical standard compliance: RS-232C

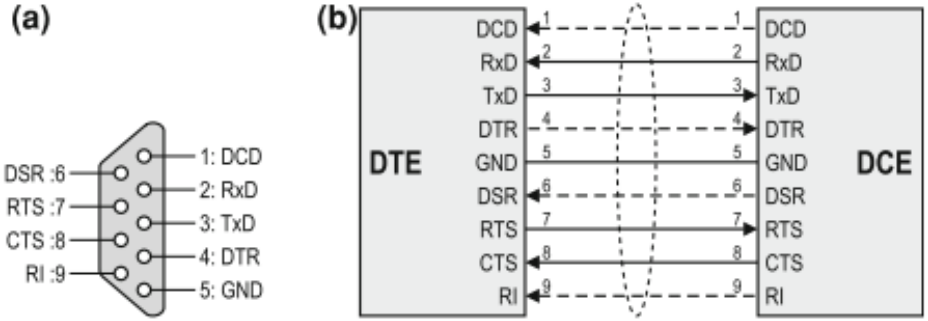


Fig. 9.12 Organization of RS-232C signals in a DE9 connector and serial cable. **a** DE-9 Connector **b** Standard serial cable

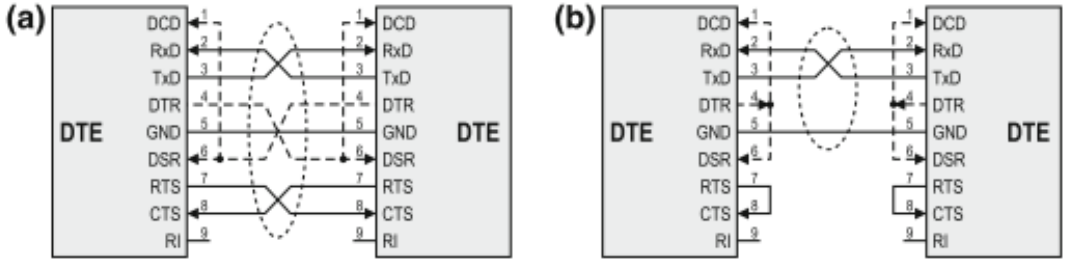


Fig. 9.13 Null-modem connection with and without handshaking. **a** Null-modem cable, **b** null-modem without handshaking

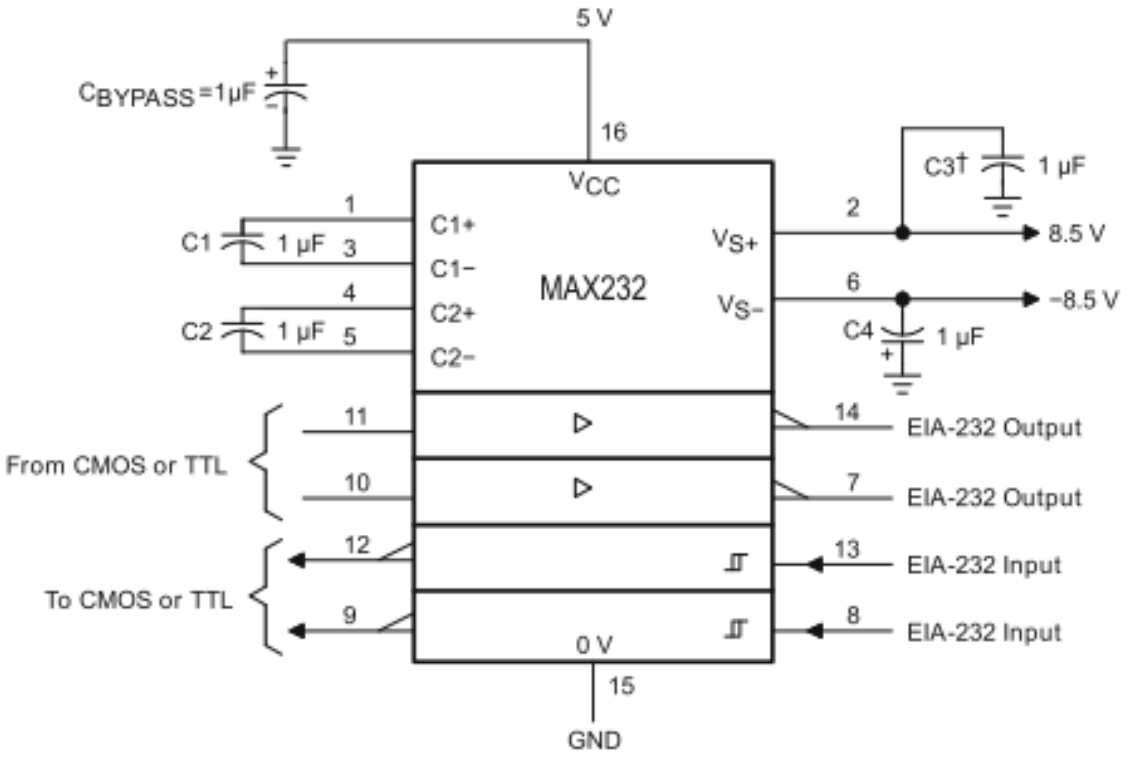


Fig. 9.14 Typical connection of a MAX232 driver/receiver (*Courtesy of Texas Instruments, Inc.*)

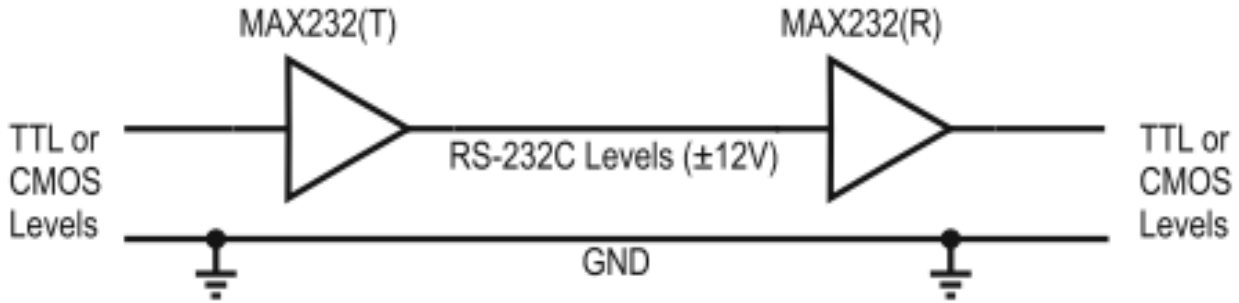


Fig. 9.15 Single-ended RS232C line

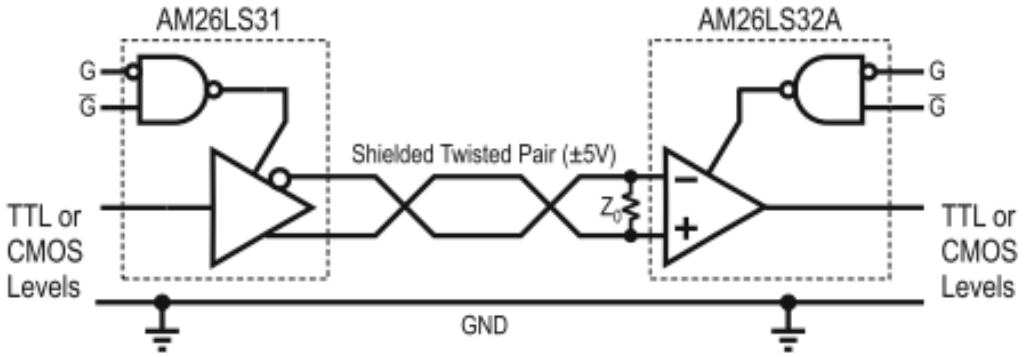


Fig. 9.16 Differential RS-422 line with AM26LS31/AM26LS32A driver/ receiver pair

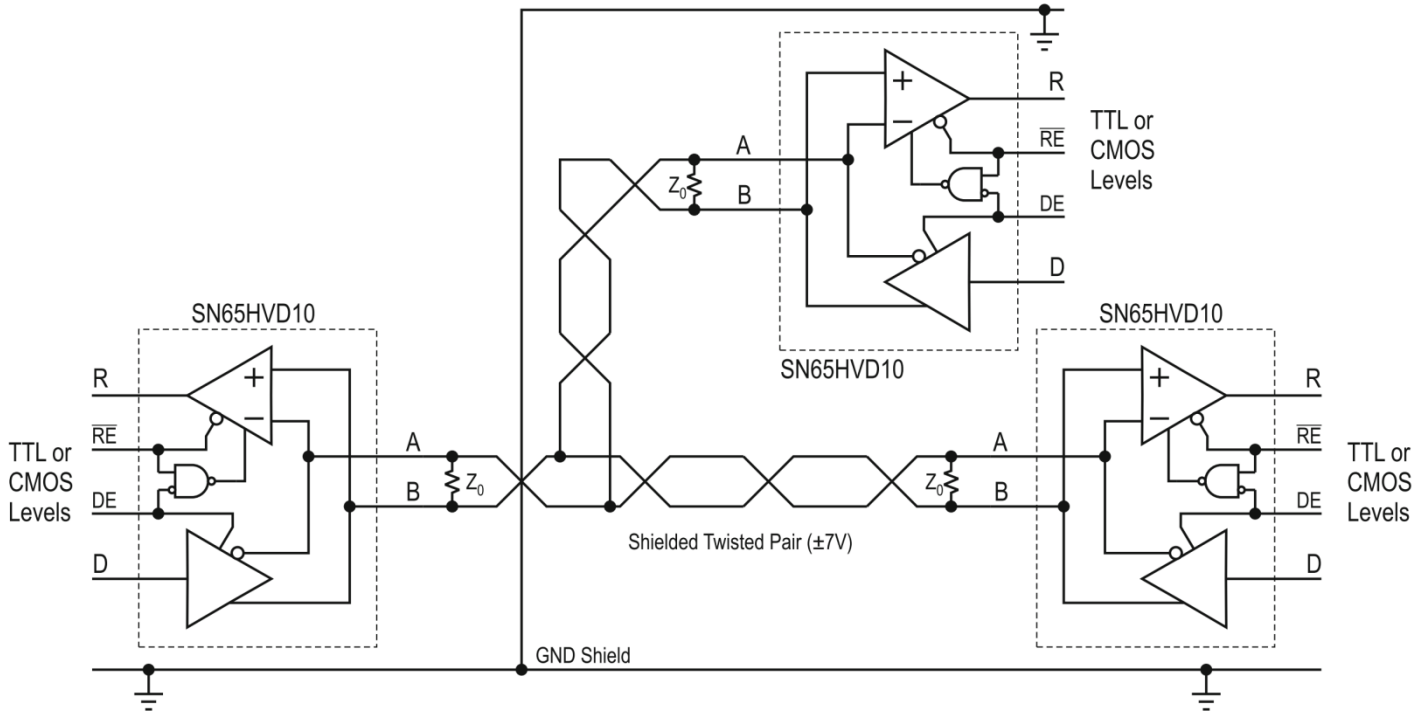


Fig. 9.17 Differential RS-485 line denoting its multidrop capability

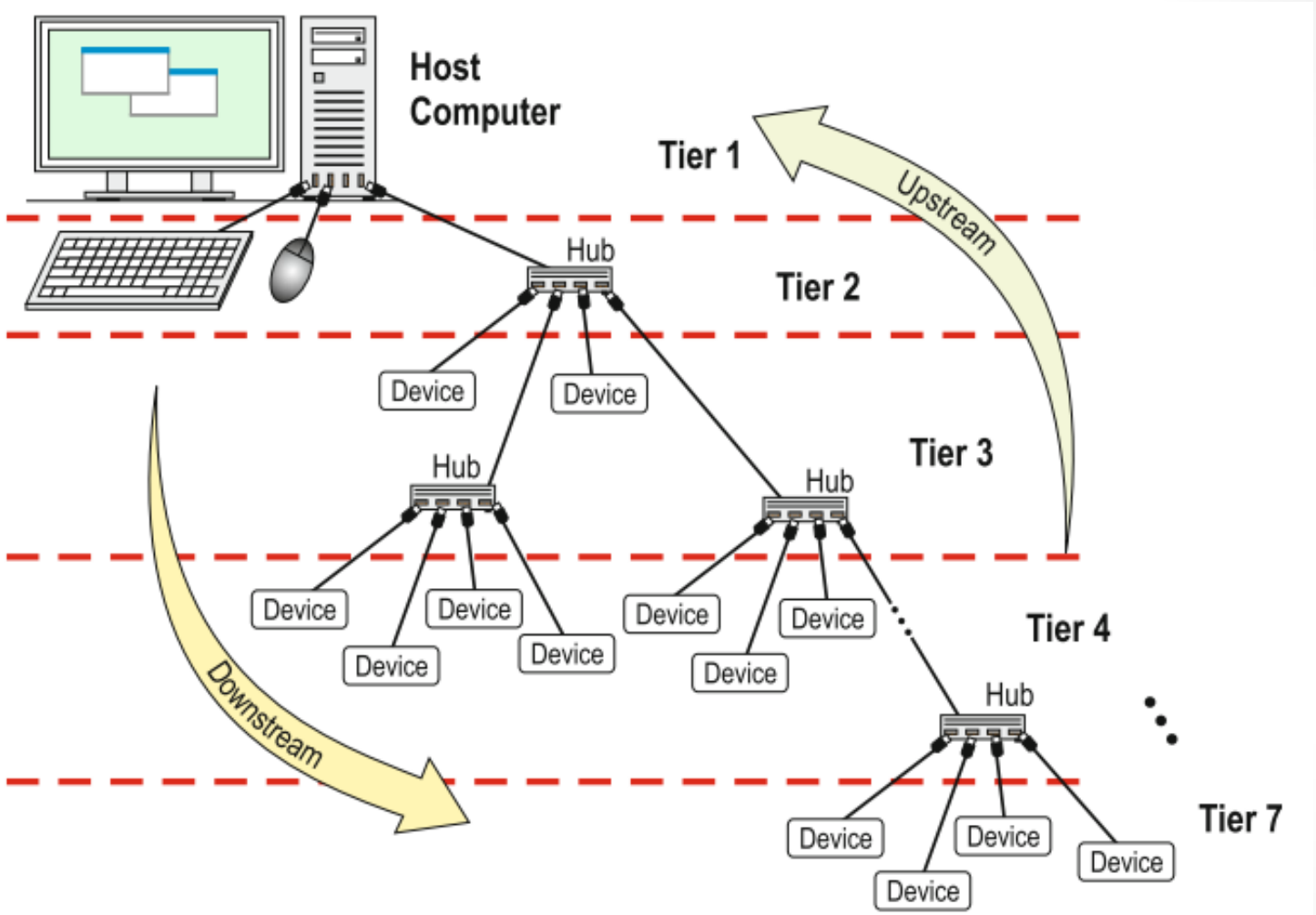


Fig. 9.18 Tiered structure of a Universal Serial Bus

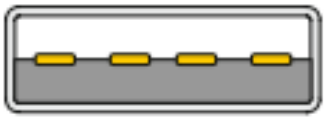
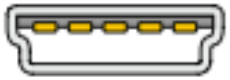
Connector	Standard	Mini	Micro
A			
B			

Fig. 9.19 Connectors in USB 2.0 cables

Table 9.1 Signal assignment in USB 2.0 connectors

Pin No.	Standard	Mini	Micro
1	$V_{Bus} = 5\,V$	$V_{Bus} = 5\,V$	$V_{Bus} = 5\,V$
2	D−	D−	D−
3	D+	D+	D+
4	GND	NC	NC
5	N/A	GND	GND

- For more details, refer to:
  - Chapter 3,8,9 at **Introduction to Embedded Systems,** Springer 2014 by Manuel Jiménez et al.

- The lecture is available online at:
  - *http://bu.edu.eg/staff/ahmad.elbanna-courses*

- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg