SERIAL BUS
COMMUNICATION PROTOCOLS
($I^2C$, CAN and USB)
1. Interconnecting number of device circuits,

- **I²C (Inter-Integrated Circuit)**, pronounced *I*-squared-*C*, is a multi-master, multi-slave, single-ended, serial computer bus invented by Philips Semiconductor (now NXP Semiconductors). It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers. ICs mutually network through a common synchronous serial bus I²C.

- An *'Inter Integrated Circuit'* (I²C) is a serial bus for interconnecting the ICs.
- I²C Bus communication—use of only simplifies the number of connections and provides a common way (protocol) of connecting different or same type of I/O devices using synchronous serial communication.
- Any device that is compatible with a I²C bus can be added to the system (assuming an appropriate device driver program is available), and a I²C device can be integrated into any system that uses that I²C bus.
Distributed Systems (ICs) on I²C Bus using serial data line and clock

- Integrated Circuit 1
  - Data
  - Clocks
  - Clock

- Integrated Circuit 2
  - Data

- Integrated Circuit 3
  - Data
I²C Bus

• The Bus has two lines that carry its signals—one line is for the clock and one is for bi-directional data.

• There is a standard protocol for the I²C bus.
Device Addresses and Master in the I2C bus

- Each device has a 7-bit address using which the data transfers take place.
- Master can address 127 other slaves at an instance.
- Master has at a processing element functioning as bus controller or a microcontroller with I2C (Inter Integrated Circuit) bus interface circuit.
• The before mentioned reference design is a bus with a **clock** and **data** lines with 7-bit addressing. The bus has two roles for nodes: master and slave:
  – Master node — node that generates the clock and initiates communication with slaves
  – Slave node — node that receives the clock and responds when addressed by the master
• The bus is a **multi-master** bus which means any number of master nodes can be present. Additionally, master and slave roles may be changed between messages (after a **STOP** is sent).
• There may be four potential modes of operation for a given bus device, although most devices only use a single role and its two modes:
  – master transmit — master node is sending data to a slave
  – master receive — master node is receiving data from a slave
  – slave transmit — slave node is sending data to the master
  – slave receive — slave node is receiving data from the master
Disadvantage of I\textsuperscript{2}C bus

- Time taken by algorithm in the hardware that analyses the bits through I\textsuperscript{2}C in case the slave hardware does not provide for the hardware that supports it.
- Certain ICs support the protocol and certain do not.
2. CAN Bus

- **CAN bus** (for **controller area network**) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other *within a vehicle without a host computer*.

- CAN bus is a message-based protocol, designed specifically for automotive applications but now also used in other areas such as aerospace, maritime, industrial automation and medical equipment.

- Development of the CAN bus started in 1983 at **Robert Bosch GmbH**. The protocol was officially released in 1986 at the **Society of Automotive Engineers** (SAE) congress in Detroit, Michigan. The first CAN controller chips, produced by Intel and Philips, came on the market in 1987.
Applications

Automotive

The modern automobile may have as many as 70 electronic control units (ECU) for various subsystems. Typically the biggest processor is the engine control unit. Others are used for transmission, airbags, antilock braking/ABS, cruise control, electric power steering, audio systems, power windows, doors, mirror adjustment, battery and recharging systems for hybrid/electric cars, etc. Some of these form independent subsystems, but communications among others are essential. A subsystem may need to control actuators or receive feedback from sensors. The CAN standard was devised to fill this need.

Industrial

Today the CAN bus is also used as a fieldbus in general automation environments, primarily due to the low cost of some CAN controllers and processors.
Serial IO CAN bus in automobile

1 Mbps multicast, multi-master, auto retransmission of erroneous data, 120 Ohm line impedance
Twisted pair CAN-H and CAN-L wires Serial IO bus

- CAN controller
- Embedded controller system B (engine controller)
- CAN controller
- Embedded controller system C (anti-lock brake controller)
- CAN controller
- Embedded controller system D (dash board)
- CAN controller
- Embedded controller system E (transmission)
3. Universal Serial Bus (USB)

- USB was designed to standardize the connection of computer peripherals (including keyboards, pointing devices, digital cameras, printers, portable media players, disk drives and network adapters) to personal computers, both to communicate and to supply electric power.

- It has become commonplace on other devices, such as smartphones, PDAs and video game consoles.

- USB has effectively replaced a variety of earlier interfaces, such as serial and parallel ports, as well as separate power chargers for portable devices.

- Variations like USB 1.X, USB 2.X, USB 3.X
Serial USB bus in a computer

- Computer system A
  - USB host Controller
    - Node
      - Camera
      - Pen like memory device
      - Printer
      - USB host Controller
    - Hub
      - Node
      - Printer
    - Hub
      - Node
      - Mobile phone
PARALLEL BUS
COMMUNICATION PROTOCOLS
(ISA, PCI and PCI/X)
Introduction

• Parallel bus enables a host computer or system to communicate simultaneously 32-bit or 64-bit with other devices or systems, for example, to a network interface card (NIC) or graphic card.
1. ISA Bus

• **Industry Standard Architecture**

• **History**
  – Originally introduced in the IBM PC (1981) as an 8 bit expansion slot
    • Runs at 8.3 MHz with data rate of 7.9 Mbytes/s
  – 16-bit version introduced with the IBM PC/AT
    • Runs at 15.9 MHz with data rate of 15.9 Mbytes/s (?)
    • Sometimes just called the “AT bus”
  – Today, all ISA slots are 16 bit

• **Configuration**
  – Parallel
• Used for...
  – Just about any peripheral (sound cards, disk drives, etc.)
• PnP ISA
  – In 1993, Intel and Microsoft introduced “PnP ISA”, for plug-and-play ISA
  – Allows the operating system to configure expansion boards automatically
• Form factor
  – Large connector in two segments
  – Smaller segment is the 8-bit interface (36 signals)
  – Larger segment is for the 16-bit expansion (62 signals)
  – 8-bit cards only use the smaller segment
• Advancements
  – EISA
    • Extended ISA
    • Design by nine IBM competitors (AST, Compaq, Epson, HP, NEC, Olivetti, Tandy, WYSE, Zenith)
    • Intended to compete with IBM’s MCA
    • EISA is hardware compatible with ISA
  – MCA
    • Micro Channel Architecture
    • Introduced by IBM in 1987 as a replacement for the AT/ISA bus
  – EISA and MCA have not been successful!
2. PCI Bus

• PCI stands for *Peripheral Component Interconnect*.

• This bus is made by Intel.

• It is used today in all PCs and other computers for connecting adapters, such as network-controllers, graphics cards, sound cards etc.

• The PCI bus is the central I/O bus, which you find in all PCs!
• PCI is a computer bus for attaching hardware devices in a computer.

• These devices can take either the form of an integrated circuit fitted onto the motherboard itself or an expansion card that fits into a slot.

• Typical PCI cards used in PCs include: network cards, sound cards, modems, extra ports such as USB or serial, TV tuner cards and disk controllers.
• **Peripheral Component Interconnect**
  – Also called “Local Bus”

• **History**
  – Developed by Intel (1993)
  – Very successful, widely used
  – Much faster than ISA
  – Gradually replacing ISA

• **Configuration**
  – Parallel
• Used for...
  – Just about any peripheral
  – Can support multiple high-performance devices
  – Graphics, full-motion video, SCSI, local area networks, etc.

• Specifications
  – 64-bit bus capability
  – Usually implemented as a 32-bit bus
  – Runs at 33 MHz or 66 MHz
  – At 33 MHz and a 32-bit bus, data rate is 133 Mbytes/s
• **PCI-X**, short for **Peripheral Component Interconnect eXtended**, is a computer bus and expansion card standard that enhances the 32-bit PCI Local Bus for higher bandwidth demanded by servers.

• It is a double-wide version of PCI, running at up to four times the clock speed, but is otherwise similar in electrical implementation and uses the same protocol.
TIMING AND COUNTING DEVICES
Timer

• Timer is a device, which counts the input at regular interval ($\delta T$) using clock pulses at its input.

• The counts increment on each pulse and store in a register, called count register.

• It has output bits for the period of counts.
Evaluation of Time

• The counts multiplied by the interval $\delta T$ give the time.

• The $(\text{present counts} - \text{initial counts}) \times \delta T$ interval gives the time interval between two instances when present count bits are read and initial counts were read or set.
Timer

- Has an input pin (or a control bit in control register) for resetting it for all count bits = 0s.
- Has an output pin (or a status bit in status register) for output when all count bits = 0s after reaching the maximum value, which also means after timeout or overflow.
Counter

- A device, which counts the input due to the events at irregular or regular intervals.
- The counts gives the number of input events or pulses since it was last read.
- Has a register to enable read of present counts
- Functions as timer when counting regular interval clock pulses
Timer cum Counting Device

• A timer cum counting device is a counting device that has two functions:
  – It counts the input due to the events at irregular instances and
  – It counts the clock input pulses at regular intervals

• A status bit in the timing device register controls the mode as Timer or Counter.
# Uses of a Timer

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Applications and Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Real Time Clock Ticks (functioning as system heart beats). [Real time clock is a clock that once the system starts it, does not stop and can’t be reset. Its count value can’t be reloaded. Real time endlessly flows and never returns!] Real Time Clock is set for ticks using prescaling bits and rate-set bits in appropriate control registers. Section 3.8 gives the details.</td>
</tr>
<tr>
<td>2.</td>
<td>Initiating an event after a preset delay time. Delay is as per count-value loaded.</td>
</tr>
<tr>
<td>3.</td>
<td>Initiating an event (or a pair of events or a chain of events) after a comparison between the preset time with counted value. Preset time is loaded in a Compare Register. [It is similar to presetting an alarm.]</td>
</tr>
<tr>
<td>4.</td>
<td>Capturing the count-value at the timer on an event. The information of time (instance of the event) is thus stored at the capture register.</td>
</tr>
<tr>
<td>5.</td>
<td>Finding the time interval between two events. Counts are captured at each event in the capture register and read. The intervals are thus found out. A service routine does the counts read on interrupt.</td>
</tr>
<tr>
<td>6.</td>
<td>Wait for a message from a queue or mailbox or semaphore for a preset time when using an RTOS. There is a predefined waiting period before RTOS lets a task run without waiting for the message.</td>
</tr>
<tr>
<td>S.No.</td>
<td>Applications and Explanation</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>7.</td>
<td>Watchdog timer. It resets the system after a defined time. Section 3.7 gives details.</td>
</tr>
<tr>
<td>8.</td>
<td>Baud or Bit Rate Control for serial communication on a line or network. Timer timeout interrupts define the time of each baud.</td>
</tr>
<tr>
<td>9.</td>
<td>Input pulse counting when using a timer, which is ticked by giving non-periodic inputs instead of the clock inputs. The timer acts as a counter if, in place of clock inputs, the inputs are given to the timer for each instance to be counted.</td>
</tr>
<tr>
<td>10.</td>
<td>Scheduling of various tasks. A chain of software-timer interrupts and RTOS uses these interrupts to schedule the tasks.</td>
</tr>
<tr>
<td>11.</td>
<td>Time slicing of various tasks. A multitasking or multiprogrammed operating system presents the illusion that multiple tasks or programs are running simultaneously by switching between programs very rapidly, for example, after every 16.6 ms. This process is known as context switch. RTOS switches after preset time-slice from one running task to the next. Each task can therefore run in predefined slots of time.</td>
</tr>
<tr>
<td>12.</td>
<td>Time division multiplexing (TDM). Timer device is used for multiplexing the input from a number of channels. Each channel input is allotted a distinct and fixed-time slot to get a TDM output. [For example, multiple telephone calls are the inputs and TDM device generates the TDM output for launching it into the optical fibre.]</td>
</tr>
<tr>
<td>S. No.</td>
<td>States</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1.</td>
<td>Reset State (initial count equals 0)</td>
</tr>
<tr>
<td>2.</td>
<td>Initial Load State (initial count loaded)</td>
</tr>
<tr>
<td>3.</td>
<td>Present State (counting or idle or before start or after overflow or overrun)</td>
</tr>
<tr>
<td>4.</td>
<td>Overflow State (count received to make count equal 0 after reaching the maximum count)</td>
</tr>
<tr>
<td>5.</td>
<td>Overrun State (several counts received after reaching the overflow state)</td>
</tr>
<tr>
<td>6.</td>
<td>Running (Active) or Stop (Blocked) state</td>
</tr>
<tr>
<td>7.</td>
<td>Finished (Done) state (stopped after a preset time interval or timeout)</td>
</tr>
<tr>
<td>8.</td>
<td>Reset enabled/disabled State (enabled resetting of count equal 0 by an input)</td>
</tr>
<tr>
<td>9.</td>
<td>Load enabled/disabled State (reset count equals initial count after the timeout)</td>
</tr>
<tr>
<td>10.</td>
<td>Auto Re-Load enabled/disabled State (enabled count equals initial count after the timeout)</td>
</tr>
<tr>
<td>11.</td>
<td>Service Routine Execution enable/disable State (enabled after timeout or overflow)</td>
</tr>
</tbody>
</table>