KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE-21

Faculty of Engineering

Department of Electronics and Communication Engineering

Lecture plan

NAME OF THE STAFF: Dr.N.Rajalakshmi

DESIGNATION : ASSISTANT PROFESSOR

CLASS : B.E-IV YEAR ECE

SUBJECT : EMBEDDED SYSTEMS

SUBJECT CODE : 14BEECE19

S.No.	TOPICS TO BE COVERED	TIME DURATION	TEACHING AIDS
UNIT-I INTRODUCTION TO EMBEDDED SYSTEMS			
1	Introduction	1	T1-p.no 3-7
2	Definition and Classification	1	T1-p.no 52
3	Overview of Processors and hardware units in an embedded system	2	T1-p.no 10-19
4	Software embedded into the system	2	T1-p.no 19-23
5	Exemplary Embedded Systems – Embedded Systems on a Chip (SoC)	1	T1-p.no 29-30
6	use of VLSI designed circuits	1	T1-p.no 30-32
7	Tutorial	1	
	Total (Theory + Tutorial)	**9Hrs (8+1)**	
UNIT II DEVICES AND BUSES FOR DEVICES NETWORK			
1	I/O Devices - Device I/O Types and Examples	1	T1-p.no 130
2	Synchronous - Iso-synchronous and Asynchronous Communications from Serial Devices	1	T1-p.no 131-134
3	Examples of Internal Serial-Communication Devices - UART and HDLC	1	T1-p.no 134-142
4	Parallel Port Devices	1	T1-p.no 142-149
5	Sophisticated interfacing features in Devices/Ports	1	T1-p.no150-151
6	Timer and Counting Devices - 'I2C'	1	T1-p.no151-156
7	'USB', 'CAN'	1	T1-p.no161-164
8	advanced I/O Serial high speed buses-ISA, PCI	1	T1-p.no165-167
9	PCI-X, cPCI and advanced buses	1	T1-p.no168-172
	Total	**9 Hrs**	
UNIT III PROGRAMMING CONCEPTS AND EMBEDDED PROGRAMMING IN C, C++			
1	Programming in assembly language (ALP) vs. High Level Language	1	T1-p.no235-237

2	C Program Elements, Macros and functions -Use of Pointers - NULL Pointers	1	T1-p.no237-244
3	Use of Function Calls – Multiple function calls in a Cyclic Order in the Main Function Pointers	1	T1-p.no 256-259
4	Function Queues and Interrupt Service Routines Queues Pointers	1	T1-p.no 259-260
5	Concepts of EMBEDDED PROGRAMMING in C++	1	T1-p.no260-261
6	Objected Oriented Programming	1	T1-p.no 262-264
7	Embedded Programming in C++	1	T1-p.no265
8	'C' Program compilers – Cross compiler Optimization of memory codes	1	T1-p.no266-271
9	Tutorial	1	
	Total (Theory + Tutorial)	**9Hrs (8+1)**	
UNIT IV REAL TIME OPERATING SYSTEMS – PART - 1			
1	Definitions of process, tasks and threads – Clear cut distinction between functions – ISRs and tasks by their characteristics	1	T1-p.no 351-353
2	Operating System Services- Goals – Structures- Kernel - Process Management	1	T1-p.no 356-360
3	Memory Management – Device Management	1	T1-p.no 360-364
4	File System Organization and Implementation – I/O Subsystems – Interrupt Routines Handling in RTOS	1	T1-p.no 364-369
5	REAL TIME OPERATING SYSTEMS : RTOS Task scheduling models - Handling of task scheduling and latency and deadlines as performance metrics	1	T1-p.no 385-388
6	Co-operative Round Robin Scheduling – Cyclic Scheduling with Time Slicing (Rate Monotonics Co-operative Scheduling)	1	T1-p.no 388-391
7	Preemptive Scheduling Model strategy by a Scheduler	1	T1-p.no 392-394
8	Critical Section Service by a Preemptive Scheduler	1	T1-p.no 394-397
9	Tutorial	1	
	Total (Theory + Tutorial)	**9 Hrs (9+1)**	
UNIT V REAL TIME OPERATING SYSTEMS – PART - 2			
1	INTER PROCESS COMMUNICATION AND SYNCHRONISATION – Shared data problem – Use of Semaphore(s)	1	T1-p.no 331-334
2	Priority Inversion Problem and Deadlock Situations	1	T1-p.no 328-330
3	Inter Process Communications using Signals	1	T1-p.no 460-464
4	Semaphore Flag or mutex as Resource key – Message Queues	1	T1-p.no 465-466
5	Mailboxes – Pipes – Virtual (Logical)	1	T1-p.no 441-449

	Sockets – Remote Procedure Calls (RPCs)		
6	Study of Micro C/OS-II or Vx Works or Any other popular RTOS	1	T1-p.no 452-456
7	RTOS System Level Functions – Task Service Functions – Time Delay Functions	1	T1-p.no 422-423
8	Memory Allocation Related Functions – Semaphore Related Functions	1	T1-p.no 424-425
9	Mailbox Related Functions – Queue Related Functions	1	T1-p.no 437-440,445
10	Tutorial	1	
	Total (Theory + Tutorial)	**9Hrs (9+1)**	

Total Lecture: 46 Hours (42+4)

TEXTBOOK:

S.NO.	Author(s) Name	Title of the book	Publisher	Year of Publication
1.	Rajkamal	Embedded Systems Architecture, Programming and Design	TATA McGraw-Hill, New York	2003

REFERENCES:

S.NO.	Author(s) Name	Title of the book	Publisher	Year of Publication
1.	Steve Heath	Embedded Systems Design	Newnes	2003
2.	David E.Simon	An Embedded Software Primer	Pearson Education Asia, New York	2000
3.	Wayne Wolf	Computers as Components : Principles of Embedded Computing System Design	Harcourt India, Morgan Kaufman Publishers, First Indian Reprint	2001
4.	Frank Vahid and Tony Givargis	Embedded Systems Design	A unified Hardware / Software Introduction, John Wiley	2002

FACULTY IN-CHARGE **HOD/ECE**

UNIT 1

Introduction to Embedded Systems

System:

A way of working, organizing or performing one or many tasks according to a fixed set of rules, program or plan.

Examples of Systems

1.Time display system – A watch

2. Automatic cloth washing system – A washing machine

Embedded System Definitions:

1. "An embedded system is a system that has software embedded into computer-hardware, which makes a system dedicated for an application (s) or specific part of an application or product or part of a larger system."

2. "It is any device that includes a "It is any device that includes a programmable computer but is not itself intended to be a general purpose computer." –***Wayne Wolf, Ref: 61***

3. "Embedded Systems are the electronic systems that contain a microprocessor or a microcontroller, but we do not think of them as computers– the computer is hidden or embedded in the system." – ***Todd D. Morton,Ref: 38***

4. "An embedded system is one that has a dedicated purpose software embedded in a computer hardware."

Three main embedded components:

1. Embedded Hardware
2. Application Software
3. RTOS

Sophisticated Embedded System Characteristics

(1) Dedicated functions
(2) Dedicated complex algorithms
(3) Dedicated (GUIs) and other user interfaces for the application
(4) Real time operations
(5) Multi-rate operations

Constraints of an Embedded System Design

Available system-memory
Available processor speed
Limited power dissipation

System design constraints

- ✓ Performance
- ✓ power
- ✓ size
- ✓ non-recurring design cost
- ✓ manufacturing costs.

Classification of Embedded Systems:

We can classify embedded systems into three types as follows l. Small Scale Embedded System

2. Medium Scale Embedded Systems
3. Sophisticated Embedded Systems

	Small Scale Embedded Systems	Medium Scale Embedded Systems	Sophisisticated Embedded Systems
SIZE	8\16bit microcontroller	Single\Few 16 or 32 bit microcontroller	Need several IP's, ASIPs,Scalable\Configurable Processors
Complexity	Little hardware and software complexity	Both hardware and software complexity	Enormous hardware and software complexity
Features	Battery Operated, Little Power Dissipation, C languages Used for Compilation.	Software tools provides solution for hardware complexity.	Consrained by processing Speeds and performs functions such as encryption, Decrption Algorithms, Discrete Cosine Transform and inverse transformation algorithms, TCP\IP protocol stacking and network driver Functions
Software Tools	Editor,Assembler, Cross Assembler	C\C++\Visual C++, Java, RTOS, Source Code Engineering Tool, Simulator, Debugger, IDE	Retargetable Compiler is needed.

I. Small Scale Embedded Systems

- These systems are designed with a 8 bit or 16 bit microcontroller.
- They have little hardware and software complexities and involve board-level design. They may even be battery operated.
- In small scale embedded systems an editor, assembler and cross assembler, specific to the microcontroller or processor used, are the main programming tools.
- Usually, 'C' is used for developing these systems. 'C' program compilation is done into the assembly, and executable codes are then appropriately located in the system memory. The software has to fit within the memory available.
- Examples:
    1. Automatic chocolate vending machine
    2. Stepper motor controller for a robotics system
    3. Washing or cooking system

2. Medium Scale Embedded Systems

- These systems are usually designed with a single or few 16- or 32-bit microcontrollers or DSPs or Reduced Instruction Set Computers (RISCs).
- These have both hardware and software complexities. For Complex software design, there are the following Programming tools: RTOS, Source code engineering tool, Simulator, Debugger and Integrated Development Environment (IDE).

Examples:

1. Computer Networking System
2. Entertainment systems
3. Embedded firewall / Router
4. Signal tracking system

3. Sophisticated Embedded Systems

- Sophisticated embedded systems have enormous hardware and software complexities and may need scalable processors or configurable processors and programmable logic arrays.
- They are used for cutting edge applications that need Hardware and software co-design and integration in the final system; however, they are constrained by the processing speeds available in their hardware units.
- Certain software functions such as encryption and Deciphering algorithms, discrete cosine transformation and inverse transforms algorithms. TCP/IP protocol stacking and network driver functions are implemented in the hardware to obtain additional speeds by saving time. Some of the functions of the hardware resources in the Systems are also implemented by the software.
- Examples:

1. Embedded system s for wireless LAN & for convergent technology devices.
2. Security products & high speed network security, gigabit rate encryption rate products
3. Embedded system for real time video & speech

Overview of Processors

-Important unit in the embedded system hardware.

-Heart of an Embedded System.

-Processors have two essential units

1.Control Unit-Includes Fetch Unit for fetching instructions from the memory.

2.Execution Unit-Performs data transfer Operations and data conversion from one to another. It consists of ALU operations and execute the instructions.

Classification:

1. General Purpose Processor
   - 1.1 Microprocessor
   - 1.2 Embedded Microprocessor
2. Application Specific Instruction set Processor
   - 2.1 Microcontroller
   - 2.2 Embedded Microcontroller
   - 2.3 Digital Signal Processor
   - 2.4 Network processor
3. Single Purpose Processor
4. GPP or ASIP cores integrated intoeither ASIC\VLSI circuit
5. Application Specific System Processor
6. Multicore processors

System designer considerations

1.Clock frequency in MHz and processingspeed – Million Instructions Per Second (*MIPS*) or Million Floating Point Instructions Per

2.(*MFLOPS*) or*Dhrystone*– an alternate metric for measuring processing performance.

3.Processor Instructions in the Instruction set

4. Processor ability to solve the complex algorithms used in meeting the deadlines for their processing.

5.Maximum bits in operand (8 or 16 or 32) in a single arithmetic or logical operation.

6.Internal and External bus-widths in the data-path

1.1 Microprocessor

A Microprocessor is a single VLSI chip that has CPU and also has some other hardware units. The CPU is a unit that has centrally fetches and processes a set of general-purpose instructions. The CPU instructions set includes instructions for data transfer operations, ALU operations, Stack operation, IO operations and program control, Sequencing and Supervising operations.

1. A control unit fetches and control the sequential processing of a given command or instruction and communicates with the rest of the system.
2. An ALU undertakes arithmetic and logical operations on bytes.

For example, Intel 80x86, Sparc, or Motorola 68HCxxx.

1.2Embedded general purpose processor

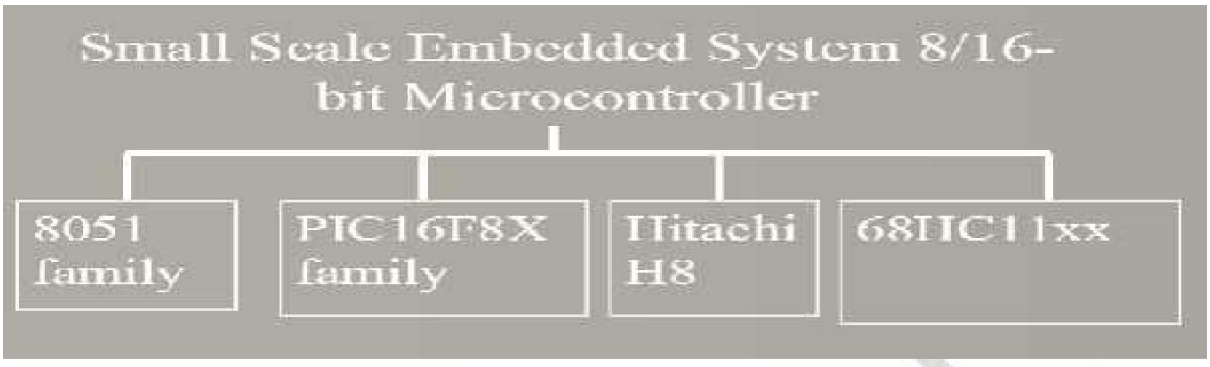
It contains the features

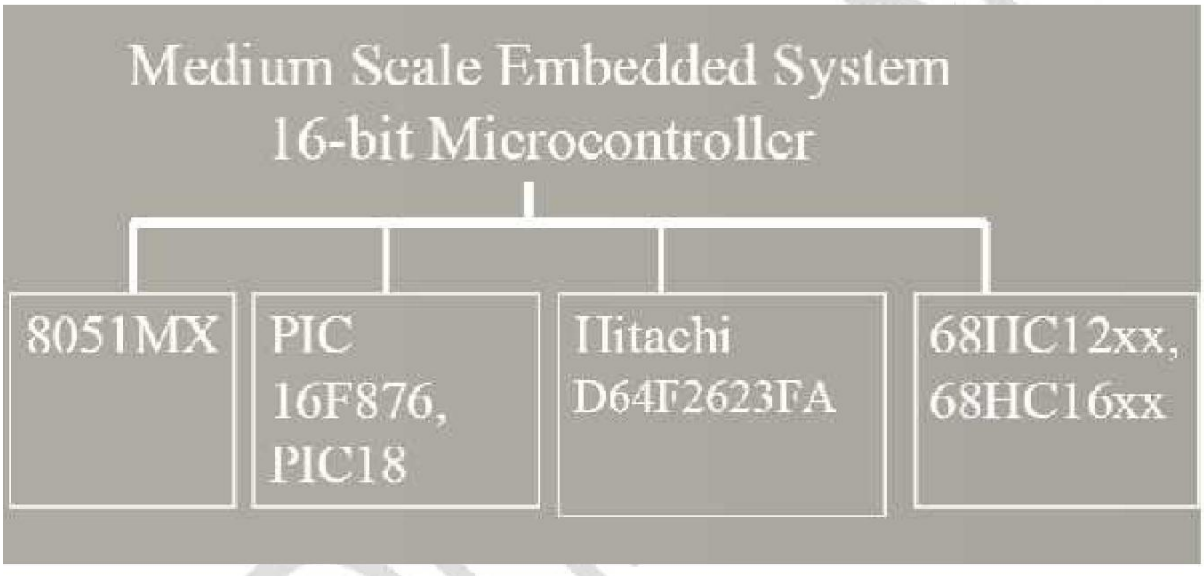
1. Fast context switching-use of on-chip Compilers, for example, Intel® XScale™ Applications Personal Internet Client Architecture-based PDAs, cell phones and other wireless devices.
2. 32-bit or 64-bit Atomic ALU operations to avoid shared data problem.
3. 32-bit Fast RISC Processor.

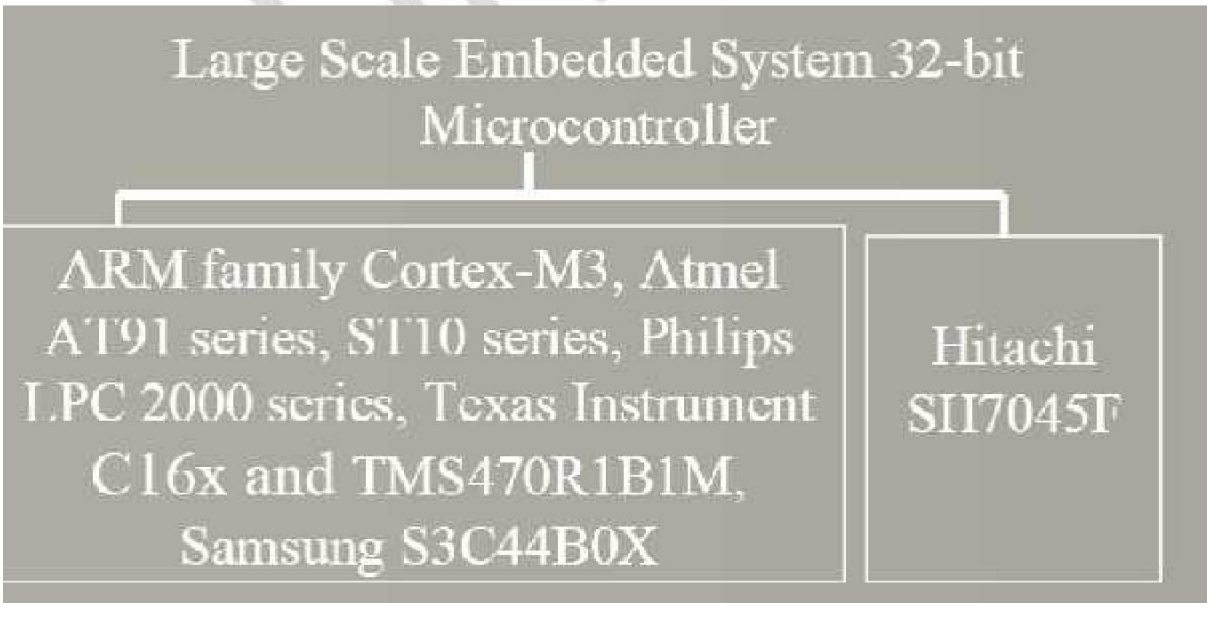
2.1MICROCONTROLLER

A Microcontroller is an integrated chip that has processor, memory,and several other hardware units. A Microcontroller is a Single-Chip VLSI unitwhich though have limited computational capabilities, Possesses enhanced input output capabilities and a number of on-chip functional units. Figure below shows Functional Circuits present in a Microcontroller.

Examples of Embedded Systems:







2.2 Embedded Microcontroller

It contains the features

1. Fast context switching-use of on-chip Compilers, for example, Intel® XScale™ Applications Personal Internet Client Architecture-based PDAs, cell phones and other wireless devices.
2. 32-bit or 64-bit Atomic ALU operations to avoid shared data problem.
3. 32-bit Fast RISC Processor.

2.3 Digital Signal Processor

Dsp is a Processor core or chip for the applications that process digital signals.DSP is a essential unit of embedded system in a large number of applications processing of signals. Typically a Texas Instruments- C28x Series, C54xx or C64xx or Analog Devices SHARC or TigerSHARC,Motorola 5600xx.

APPLICATIONS:

Filtering, Noise Cancellation, Echo Elimination, Compression and Encryption Algorithm.

3SINGLE PURPOSE PROCESSORS

Used for specific purpose Embedded Applications.

- Floating point Coprocessor
- CCD Pixel coprocessor and image codec in digital camera
- Graphic processor
- Speech processor
- Adaptive filtering processor
- Encryption engine
- Decryption engine
- Communication protocol stack processor
- Java accelerator
- CODEC-1.JPEG CODEC
  2.MPEG CODEC

4 GPP or ASIP core (s)

GPP or ASIP Integrated into either an Application Specific Integrated Circuit (ASIC), or a *Very Large Scale Integrated Circuit* (VLSI) circuit or a FPGA core integrated with processor unit(s) in a VLSI (ASIC) chip. Using VLSI design Tools GPP\ASIP with instruction set required in specific application areas can be designed.

5 APPLICATION SPECIFIC SYTEM PROCESSOR

ASSP is dedicated to these specific tasks alone provides a faster solution. ASSP is configured and interfaced with embedded system. ASSP provides a solution for the problem If software alone used in some applications it takes some longer time. In some cases ASSP chip has a TCP, UDP, IP, ARP, and ETHERNET. EXAMPLES:

Typically a set top box processor or mpeg video-processor or network application processor or mobile application processor.

6 MULTICORE PROCESSORS

Several processors\Dual core Processors may be needed to execute an algorithm fast within a strict deadline. **Examples** Multiprocessor system for Real time performance in a video-conference system, Embedded firewall cum router,High-end cell phone.

COMPONENTS OF EMBEDDED SYSTEMS

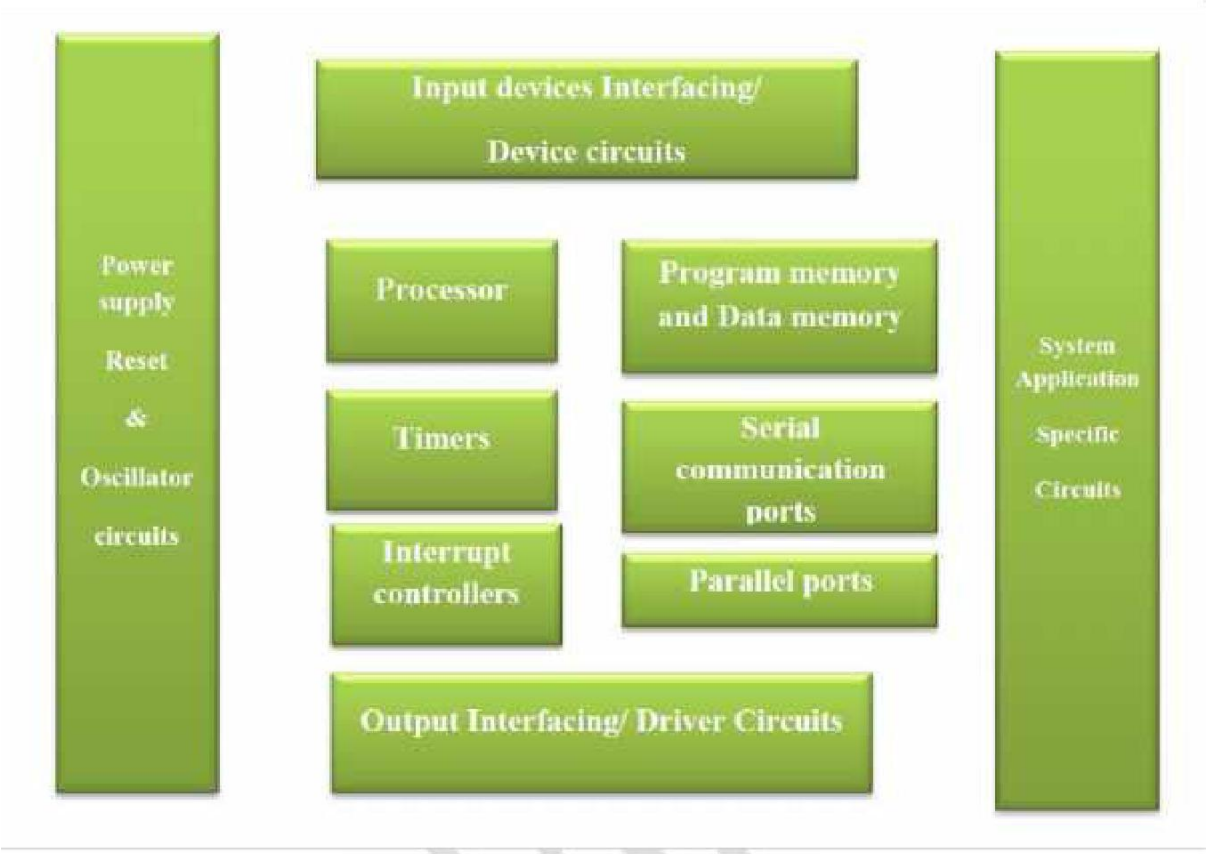


Figure: the components of embedded system

hardware Power source

Most systems have a power supply of own. The supply has a specific operation range of voltages. Various units in an embedded system operate in one of the following four operation range:

a) 5.0V±0.25V
b) 3.3V±0.3V
c) 2.0V±0.2V
d) 1.5V±0.2V

Generally, an inverse relationship between propagation delay in the gates and operational voltage. So, the 5V system processor is used most high performance systems.

Certain systems do not have a power source of their: They connect to external power supply or are powered by the use of charge pumps.

Ex: The Network Interface Card (NIC) and Graphic Accelerator.

Low voltage operations

Portable or handheld devices such as a cellular phone.

In a system with smaller overall geometry, low voltage system processors and IO circuits generate lesser heat and thus can be be packed into a smaller space.

Clock oscillator circuit and clocking units

The clock controls the time for executing an instruction. The clock controls the various clocking requirements of the CPU, of the system timers and the CPU machine cycles. The machine cycles are for

I. Fetching the codes and data from memory
II. Decoding and executing at the processor,
and III. Transferring the results to memory

For processing units, a highly stable oscillator is required and the processor clock-out signal provides the clock for synchronizing all system units with the

processor. **System timers and real time clocks**

A timer circuit is suitably configured as the **system-clock**, generates system interrupts periodically; for example, 60 times in 1s.

A timer circuit is suitably configured as the real-time clock **(RTC)** that generates system interrupts periodically for the schedulers, real-time programs and periodic saving of time and date in the system.

The RTC or system timer is also used to obtain software- controlled delays and timeouts. An RTC functions as drivers for software timers.

Reset circuit, power-up reset and watchdog-timer reset

Reset means that the processor begins the processing of instructions from a starting address. That address is one that is set by default in the processor Program Counter on a power-up. From that address in memory, program instructions are fetched following the reset of the processor.

I. A System program executes from
beginning II. A System boot-up program
III. A System initialization program
In certain processors, there are two start-up addresses.

One is based on the power-up reset vector, other on the reset vector after the reset instruction or after a time-out (watchdog timer).

On deactivation of the reset that succeeds the processor activation, a program executes from a start-up address.

Reset can be activated by

I. An external reset circuit that activates on the power-up, on switching-on reset of the system or detection of a low voltage.

II. By a software instruction or time-out by a programmed timer known as watchdog timer.

(Watchdog timer is a timing device that resets the system after a predefined timeout.)

Memory

In a system, there are various types of memories. They are as follows:

Internal RAM of 256 or 516 bytes in a microcontroller for registers, temporary data and stack.

Internal ROM/PROM/E2PROM for about 4kB to 64kB of
program. External RAM for the temporary data and stack or
internal caches. Internal flash.

Memory stick.

External ROM or PROM for embedding software.

RAM memory buffers at ports.

Caches (in pipelined and superscalar microprocessors).

Memory needed	functions
ROM or EPROM	**Storing application programs from where the processor fetches the instruction codes. Storing codes for system booting, initializing, initial input data and strings. Codes for RTOS. Pointers of various interrupt service routines (ISRs).**
RAM(internal and external) and RAM for buffer	**Storing variables during program run and storing the stack. Storing input and output buffers,**
Memory stick	**A flash memory stick is inserted in mobile computing system or digital camera. It stores high definition video, images, songs, or speeches after a suitable compression.**
EEPROM or Flash	**Storing nonvolatile results of processing**
Cache	**Storing copies of instructions and data in advance from external memories and storing results temporarily during processing.**

Input, Output and IO Ports, IO buses and IO Interfaces

The system gets inputs from physical devices through the **input ports.** Examples are as follows:

A system gets inputs from the touch screen, keypad.

A controller in a system gets input from the sensors and transducer.

A receiver of signals or a network card gets the input from a communication system.

Port receives inputs from a network or peripheral.

The system has output ports through which it sends output bytes to the real world. Examples are as follows:

Output may be sent to an LED, LCD or touch screen display panel

A system may send the output to a printer.

Output may be sent to a communication system or network

A control system sends outputs to alarms, actuators, furnaces or boilers.

Each output ports or input ports are identified by its memory –buffer addresses (called port addresses).

There are also ports for both the input and output(IO) operations. For example. Touch screen

Ports can have a serial or parallel communication with the system address and data buses. In serial communication a one-bit data line is used and bits are sent serially in successive time slots. Ex: UART. In parallel communication, several data lines are used and bits are sent in parallel.

Bus A system might have to be connected to a number of other devices and systems. For networking the systems, there are different types of buses. For example, I2C, CAN, USB, ISA, EISA and PCI. For wireless networking of systems there are 802.11, IrDA, Bluetooth and Zigbee protocols.

Interrupts Handler

A system may possess a number of devices and the system processor has to control and handle the requirements of each device by running an appropriate ISR for each.

An **interrupt handling mechanism** must exist in each system to handle interrupts from various processes and for handling multiple interrupts simultaneously pending for service.

There can be a number of **interrupt sources** and groups of interrupt sources in a processor.

1. An interrupt may be a hardware signal that indicates the occurrence of an event.

2. An interrupt may also occur through timers.

3. An interrupt may occur through an interrupting instruction of the processor program or through an error during processing.

4. An interrupt can also arise through a software timer.

The system may prioritize sources and service them accordingly.

The processor's current program has to divert to a service routine to complete that task on the occurrence of the interrupt.

There is a programmable unit on-chip for the interrupt handling mechanism in a microcontroller.

The OS is expected to control the handling of interrupts and running of routines for the interrupts in a particular application.

DAC using a PWM and an ADC

DAC is a circuit that converts digital 8 or 10 or 12 bits to the analog output. The analog output in with respect to the reference voltage.

A pulse width modulator (PWM) with an integrator circuit is used for this DAC. A PWM unit in the microcontroller operates as follows: Pulse width is made proportional to the analog-output needed.

ADC is a circuit that converts the analog input to digital 4,8,10 or 12 bits.

The ADC in the system microcontroller can be used in many applications such as data acquisition systems, digital cameras, analog control systems and voice digitizing systems.

Important points about the ADC are as follows:

1. Either a single or dual analog reference voltage source is required in the ADC. It sets either the analog input's upper limit or the lower and upper limits both. For a single reference source, the lower limit is set to 0V. When the analog input equals the lower limit, the ADC generates all bits as 0s and when it equals the upper limit it generates all bits as 1s.

2. An ADC may be of 8, 10, 12 or 16 bits depending upon the resolution needed for conversion.

3. The start of conversion (STC) signal or input initiates the conversion to 8 bits.

4. There is an end of conversion (EOC) signal that brings the conversion process to end.

5. A Sample and Hold unit is used to sample the input for a fixed time and hold till conversion is over.

LCD, LED and Touchscreen Displays

A system may need the necessary interfacing circuit and software for the output to the LCD display controller and the LED interfacing ports or for the IOs with the touchscreen.

Keypad/Keyboard

For inputs, a keypad or board may interface to a system. The system provides necessary interfacing circuit and software to receive inputs directly from the keys or through a controller.

Pulse dialer, Modem and Transceiver

For user connectivity through the telephone line, wireless or a system provides the necessary interfacing circuit.

It also provides the software for pulse dialing through the telephone line, for modem interconnection for fax, for internet packets routing, and for transmitting and connecting a WAG (wireless gateway) or cellular system.

EMBEDDED SOFTWARE IN A SYSTEM

Final machine implementable software for a product

An embedded system processor executes software that is specific to a given application of that system. The instruction codes and data in the final phase are placed in ROM or flash memory for all the tasks that are executed when the system runs. The software is called ROM image.

The image consists of boot up program, stack address pointers, program counter address pointers, application program, ISRs, RTOs, input data and vector addresses.

Each code or data is available only in bits or bytes format. The system requires bytes at each ROM address, according to the tasks being executed.

A machine implementable software file is therefore like a table having in each rows the address and bytes. The bytes are saved at each address of the system memory. The table has to be readied as a ROM image.

Figure: system ROM memory embedding the software, RTOS, data and vector addresses

Coding of software in machine codes

In configuring some specific physical device or subsystem, machine codes based coding is used. However, coding in machine implementable codes is done only in specific situations because it is time consuming.

For example:

TRANSCEIVER- placing certain machine code & bits can configure it to transmit at specific megabytes per second or gigabytes per second, using specific bus and networking protocols.

Configuring CONTROL REGISTER with the processor- during a specific code section processing, the register can be configured to enable or disable use of its internal cache.

Software in processor specific assembly language

A program or a small specific part can be coded in assembly language using an assembler after understanding the processor and its instruction is software used for developing codes in assembly.

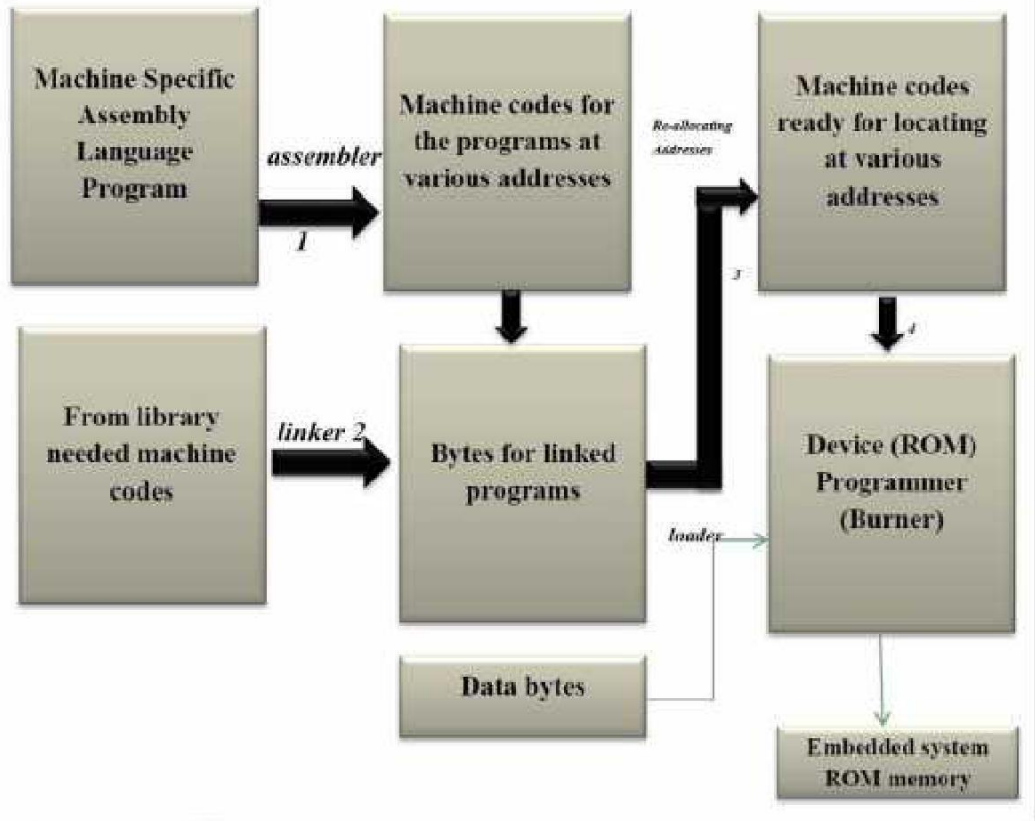


Figure: The process of converting an assembly language program into the machine codes and finally obtaining the ROM image.

Steps

1. An assembler translates the assembly software into machine codes using a step called assembling.

2. In the next step, called linking, a linker links these codes with the other required codes. The linked file in binary for run on a computer is commonly known as executable file or simply '**.exe**' file. After linking there has to be re-allocation of the sequences of placing the codes before placement of the codes in the memory.

3. In the next step, the loader program performs the task of reallocating the codes after finding the physical RAM addresses available at a given instant. The loader is a part of the operating system and places codes into the memory after reading the '.exe' file.

4. The final step of the system design process is locating the codes as a ROM image and permanently placing them at the actually available addresses in the ROM. In embedded systems, there is no separate program to keep track of the available addresses at different times during the running, as in a computer.

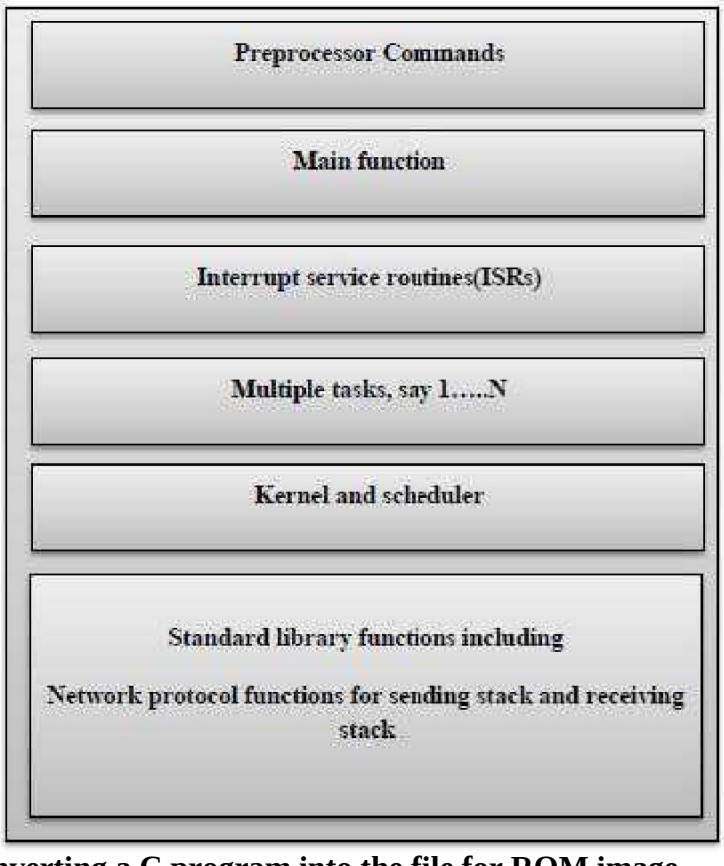
5. Lastly, either (i) a laboratory system, called device programmer, tasks as input the ROM image file and finally and finally burns the image into the PROM or flash. (ii) at a foundry, a mask is created for the ROM of the embedded system from the image file.(The process of placing the codes in PROM or EPROM is also called burning.)

Software in high level language

Since the coding in assembly language is very time consuming, software is developed in a high level language, 'C' or 'C++' or 'Java' in most cases.

'C' programs have a feature that adds the assembly instructions when using certain processor specific features and coding for a specific section, for example: a port device driver.

The **program layers** in embedded software in C:



The process of converting a C program into the file for ROM image

A compiler generates the object codes. The compiler assembles the codes according to the processor instruction set and other specifications. The 'C' compiler for embedded system must, as a final step of compilation, use a code optimizer that optimizes the codes before linking

After compilation, the linker links the object codes with other needed codes. Codes for device management and driver also link at this stage. After linking, the other steps for creating a file for ROM image are the **loader** and **locator** as in assembly language into ROM image conversion.

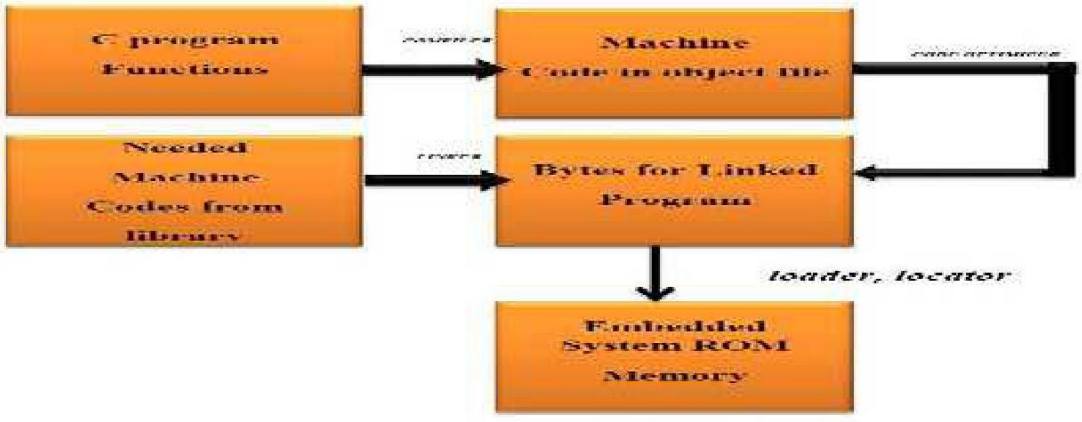


Figure: The process of converting a C program into the file for ROM image

Program Models for software designing

The program design task is simplified if a program is modeled. The different models that are employed during the design process of the embedded software as follows:

1. Sequential program model
2. Object oriented program model
3. Control and data flow graph or synchronous data flow graph or Multi Thread Graph model
4. Finite State Machine for data path.
5. Multithreaded Model for concurrent processing of processes or threads or tasks.

Software for Device drivers, Device manager using an Operating System

Devices

o In an embedded system, there are two types of devices.

o Physical devices – keypad, LCDdisplay or touch screen, memory stick(flash memory), wireless networkingdevice, parallel port and networkcard.

o Virtual devices – pipe, file, RAM disk, socket,

A *device driver* is software for controlling (configuring), receiving and sending a byte or a stream of bytes from or to a device. A set of generic functions, such as create ( ),open ( ), connect ( ), listen ( ), accept ( ), read ( ), write ( ), close ( ), delete ( ) for use by high level programmers. Each generic function calls a specific software (interrupt service routine), which controls a device function or device input or output

Device controls and functions by :

1. Calling an ISR (also called Interrupt Handler Routine) on hardware or software interrupt
2. Placing appropriate bits at the control register or word.
3. Setting status flag(s) in the status register for interrupting, therefore running (driving) the ISR, Resetting the status flag after interrupt service.

Device Manager for the devices and drivers

Device Management software (usually a part of the OS) provide codes for detecting the presence of devices, for initializing (configuring) these and for testing the devices that are present. Also includes software for allocating and registering port(s) or

device codes and data at memory addresses for the various devices at distinctly different addresses, including codes for detecting any collision between the allocated addresses, if any. An OS also provides and execute modules for managing devices for an embedded system.

Software Design for Scheduling Multiple tasks and Device using an RTOS

An embedded system is designed for scheduling multiple functions while controlling multiple devices.

□□system (OS) and Real time operating system (RTOS), Concurrent Processes, tasks or threads

□□A System is composed of two or more concurrent proce t execute Operating System

tasks

interprocessor communication, shared memory, security, GUIs, ...
management Real Time Operating System (RTOS)

An RTOS\OS has a kernel. The kernel important function is to Schedule the transition of task from ready state to Running state. The kernel coordinates the use of processor for multiple task. RTOS are highly complex. RTOS is needed when the tasks for the system have real time constraints and deadlines for finishing the tasks.

Software tools for designing an embedded system

Editor

It is used for writing C codes or assembly mnemonics using the keyboard of the PC for entering the program.

It allows the entry, addition, deletion, insert, appending previously written lines or files, merging record and files at the specific positions. It creates a source file that stores the edited file

Interpreter

It is used for expression by expression (line by line) translation to the machine executable codes.

Compiler

It uses the complete sets of the codes. It may also include the codes, functions and expressions from the library routines. It creates a file called object file.

Assembler

It translates the assembly mnemonics into binary opcodes i.e into an executable file called a binary file.

It also creates a list file that can be printed. The list file has address, source code and hexadecimal object codes. The file has addresses that adjust during the actual run of the assembly language program.

Cross assembler

For converting object codes or executable codes for a processor to other codes for another processor and vice versa. The cross assembler assembles the assembly codes of the target processor as the assembly codes of the processor of the PC used in system development. Later, it provides the object codes for the target processor.

Simulator

To simulate all functions of an embedded system circuit including that or additional memory and peripherals. It is independent of a particular target system. It also simulates the processes that will execute when the codes of a particular processor execute.

Source- code engineering software

For source code comprehension, navigation and browsing, editing, debugging, configuring (disabling and enabling the C++ features) and compiling.

Stethoscope

For dynamically tracking the changes in any program variable or parameter. It demonstrates the sequence of multiple processes that execute and also records the entire time history.

Trace scope

To help in tracing the changes in modules and tasks with time on the X-axis. A list of actions also produces the desired time scales and the time expected to be taken for different tasks.

Integrated Development Environment (IDE)

This is a development software and hardware environment that consists of simulators with editors, compilers, assemblers, RTOs, Debuggers, stethoscope, tracer, emulators, logic analyzers and application code burners in PROM.

Prototyper

This simulates and does source code engineering including compiling, debugging and browsing and summarizing the complete status f the final target system during the development phase.

Locator

This uses cross assembler output and a memory allocation map and provides the locator program output as a hex- file. It is the final step of the software design process or an embedded system.

EXAMPLES OFEMBEDDED

SYSTEMS Application Areas

o Telecom
o Smart Cards,
o Missiles and Satellites,
o Computer Networking,
o Digital Consumer Electronics, and
o Automotive

EXAMPLES

Small Scale Embedded System

o Automatic Chocolate Vending Machine
o Stepper motor controllers for a robotics system
o Washing or cooking system
o Multitasking Toys
o Microcontroller- based single or multi-display digital panel meter for voltage, current, resistance and frequency
o Keyboard controller
o Serial port cards
o CD drive or Hard Disk drive controller
o Peripheral controllers,, a CRT display controller, a keyboard controller, a DRAM controller, a DMA controller, a printer controller, or a laser printer-controller, a LAN controller,a disk drive controller
o Fax or photocopy or printer or scanner Machine Remote (controller) of
TV o Telephone with memory, display and other sophisticated features
o Motor controls Systems - for examples, an accurate control of speed and position of d.c. motor, robot, and CNC machine;, the automotive applications like such as a close loop engine control, a dynamic ride control, and an anti-lock braking system monitor

o Electronic data acquisition and supervisory control system Spectrum analyzer

Medium Scale Embedded Systems

Computer networking systems, - for examples, router, front-end processor in a server, switch, bridge, hub, and gateway

For Internet appliances, there are numerous application systems

(i) Intelligent operation, administration and maintenance router (IOAMR) in a distributed network, and

(ii) Mail Client card to store e-mail and personal addresses and to smartly connect to a modem

or server

Banking systems - for examples, Bank ATM and Credit card transactions

Signal Tracking Systems - for examples, an automatic signal tracker and a target tracker.

Communication systems, for examples, such as for a mobile-communication a SIM card, a numeric pager, a cellular phone, a cable TV terminal, and a FAX transceiver with or without a graphic accelerator. Image Filtering, Image Processing, Pattern Recognizer, Speech Processing and Video Processing.

Entertainment systems - such as videogame, music system and Video Games

A system that connects a pocket PC to the automobile driver mobile phone and a wireless receiver. The system then connects to a remote server for Internet or e-mail orto remote computer at an ASP (application Service Provider).A personal information manager using frame buffers in hand- held devices.

Thin Client to provide the disk-less nodes with the remote boot capability.[Application of thin- clients is accesses to a data center from a number of nodes; or in an Internet Laboratory accesses to the Internet leased line through a remote Server]. Embedded Firewall / Router using ARM7/multi-processor with two Ethernet interfaces and interfaces support to for PPP, TCP/IP and UDP protocols.

Sophisticated Scale Embedded Systems

es and Computing systems

version 6) Internet and other products, real time video and speech or multimedia processing systems

ultra high speed (10 Gbps) and large bandwidth: Routers, LANs, switches and gateways, SANs (Storage Area Networks), WANs (Wide Area Networks), Security products and High-speed Network security, Gigabit rate encryption rate products

EMBEDDED SYSTEM ON CHIP (SOC) AND IN VLSI CIRCUIT

Embedded systems are being designed on a single silicon chip, called System on Chip (SOC). SoC is a system on a VLSI chip that has all the necessary analog as well as digital circuits, processors, and software.

A SoC may be embedded with the following components:

1. Embedded processor GPP or ASIP core,
2. Single purpose processing cores or multiple processors,
3. A network bus protocol core,
4. An encryption function unit,
5. Memories
6. Multiple standard source solutions, called IP (Intellectual property) cores,
7. Programmable logic device and FPGA cores,
8. Digital and analog units.

Ex: mobile phone. Single purpose processors, ASIPs and IPs on an embedded SoC are configured to process encoding, dialing, modulating, demodulating, interfacing the keypad and multiple line LCD matrix displays and touch screen, storing data input and recalling data from memory.

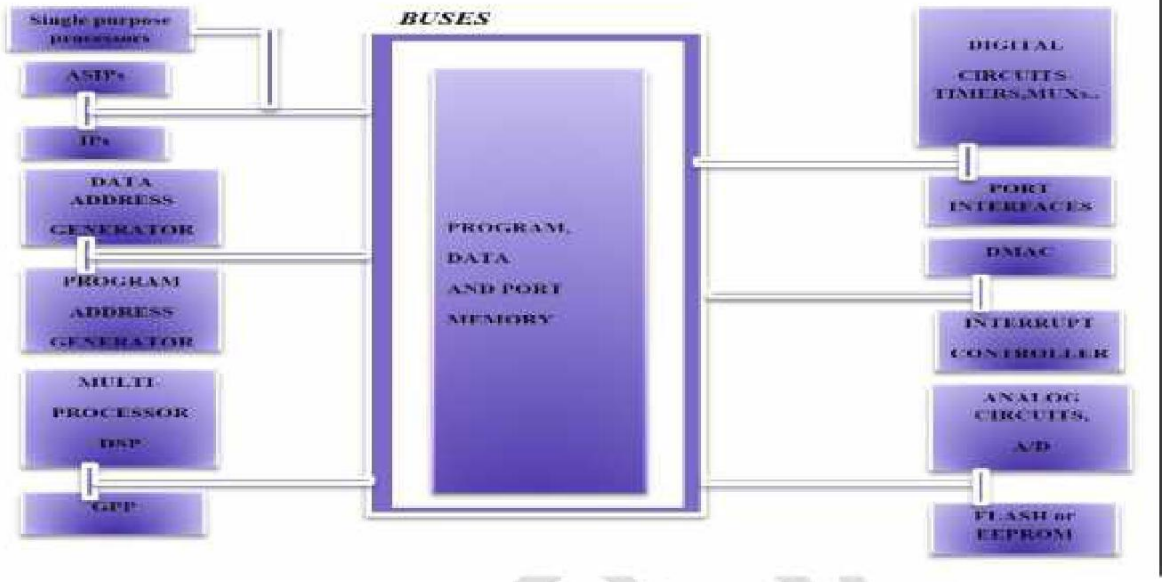


Fig: A soc embedded sytem and its common bus with two internal ASICs, two internal processors, shared memories and peripheral interfaces

Application Specific IC (ASIC)

ASICs are designed using the VLSI design tools with the processor GPP or ASIP and analog circuits embedded into the design.

The designing is done using the Electronic Design Automation (EDA) tool. For design of an ASIC, a High-level Design Language (HDL) is used.

IP Core

On a VLSI chip, there may be integration of high-level components. These components possess gate-level sophistication in circuits above that of counter, register, multiplier, floating point operation unit and ALU.

A standard source solution for synthesizing a higher level component by configuring FPGA core or a core of VLSI chip may be available as an intellectual property, called (IP)

The Copyright for the synthesized design of a higher level component for gate-level implementation of an IP is held by the designer or designing company.

1. An IP may provide hardwired implementable design of a transform, an encryption algorithm or a deciphering algorithm.
2. An IP may provide a design for adaptive filtering of a signal.
3. An IP may provide a design for implementing Hyper Text Transfer Protocol (HTTP) or File Transfer Protocol (FTP) or Bluetooth protocol to transmit a web page or a file on the Internet.
4. An IP may be designed for a USB or PCI bus controller.

FPGA Core with Single or Multiple Processors

A new innovation is Field Programmable Gate Arrays (FPGA) core with a single or multiple processor units on chip.

An FPGA consists of a large number of programmable gates on a VLSI chip. There is a set of gates in each FPGA cell, called macro cell. Each cell has several inputs and outputs and they are interconnected like an array (matrix). Each interconnection is programmable through the associated RAM in an FPGA programming tool.

Consider the algorithm for the following: an SIMD instruction, Fourier transform and its inverse, DFT or Laplace transform and its inverse, compression or decompression, encrypting or deciphering, pattern recognition. We can configure an algorithm into the logic gates of FPGA. It gives hardwired implementation for a processing unit.

One example is Xilinx Virtex-II Pro FPGA XC2VP 125. XC2VP125 from Xilinx has 125136 logic cells in the FPGA core with four IBM power PCs. It has been used as a data security solution with encryption engine and data rate of 1.5Gbps.

UNIT II

DEVICES AND BUSES FOR DEVICES NETWORK

IO port types- Serial and parallel IO ports

A port is a device to receive the bytes from external peripheral(s) [or device(s) or processor(s) or controllers] for reading them later using instructions executed on the processor to send the bytes to external peripheral or device or processor using instructions executed on processor.

A Port connects to the processor using address decoder and system buses. The processor uses the addresses of the port-registers for programming the port functions or modes, reading port status and for writing or reading bytes.

Example

- SI serial interface in 8051
- SPI serial peripheral interface in 68HC11
- PPI parallel peripheral interface 8255
- Ports P0, P1, P2 and P3 in 8051 or PA, PB,PC and PD in 68HC11
- COM1 and COM2 ports in an IBM PC

IO Port Types

Types of Serial ports

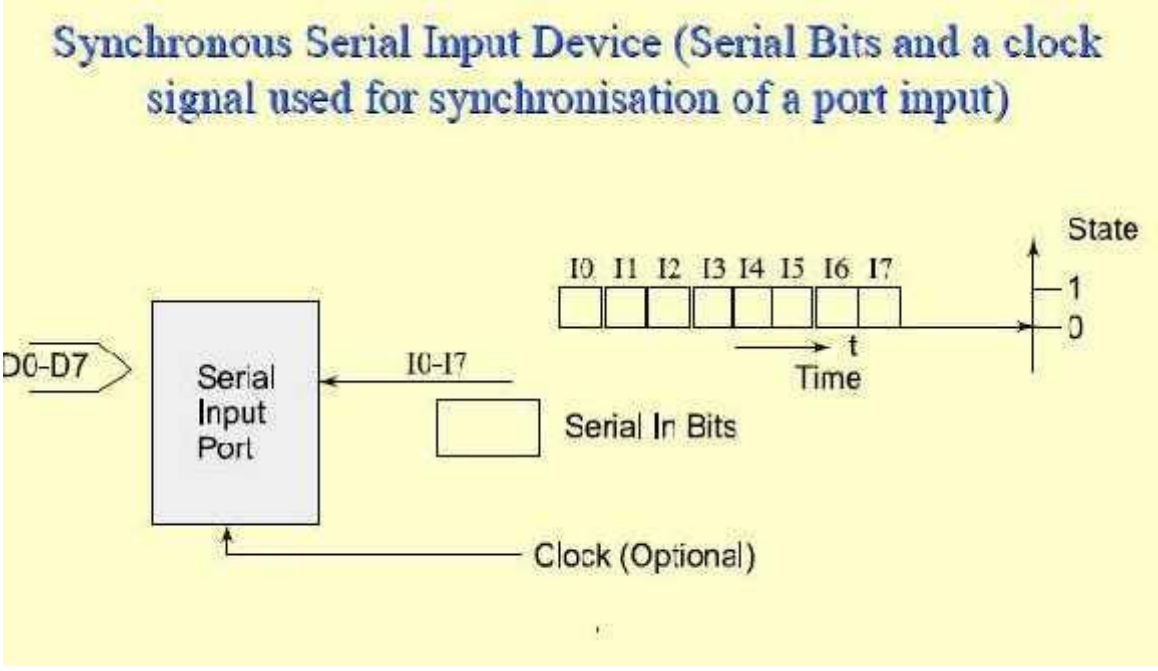
- Synchronous Serial Input
- Synchronous Serial Output
- Asynchronous Serial UART input
- Asynchronous Serial UART output (both as input and as output, for example,modem.)
-

Types of parallel ports

- Parallel port one bit Input
- Parallel one bit output
- Parallel Port multi-bit Input
- Parallel Port multi-bit Output

Synchronous Serial Input Example

Inter-processor data transfer, reading from CD or hard disk, audio input, video input, dial tone, network input, transceiver input, scanner input, remote controller input, serial I/O bus input, writing to flash memory using SDIO (Secure Data Association IO based card).



Synchronous Serial Input

- The sender along with the serial bits also sends the clock pulses SCLK (serial clock) to the receiver port pin. The port synchronizes the serial data input bits with clock bits. Each bit in each byte as well as each byte in synchronization
- Synchronization means separation by a constant interval or phase difference. If clock period = T, then each byte at the port is received at input in period = 8T.
- The bytes are received at constant rates. Each byte at input port separates by 8T and data transfer rate or the serial line bits is (1/T) bps. [1bps = 1 bit per s]
- Serial data and clock pulse-inputs
- On same input line − when clock pulses either encode or modulate serial data input bits suitably. Receiver detects the clock pulses and receives data bits after decoding or demodulating.

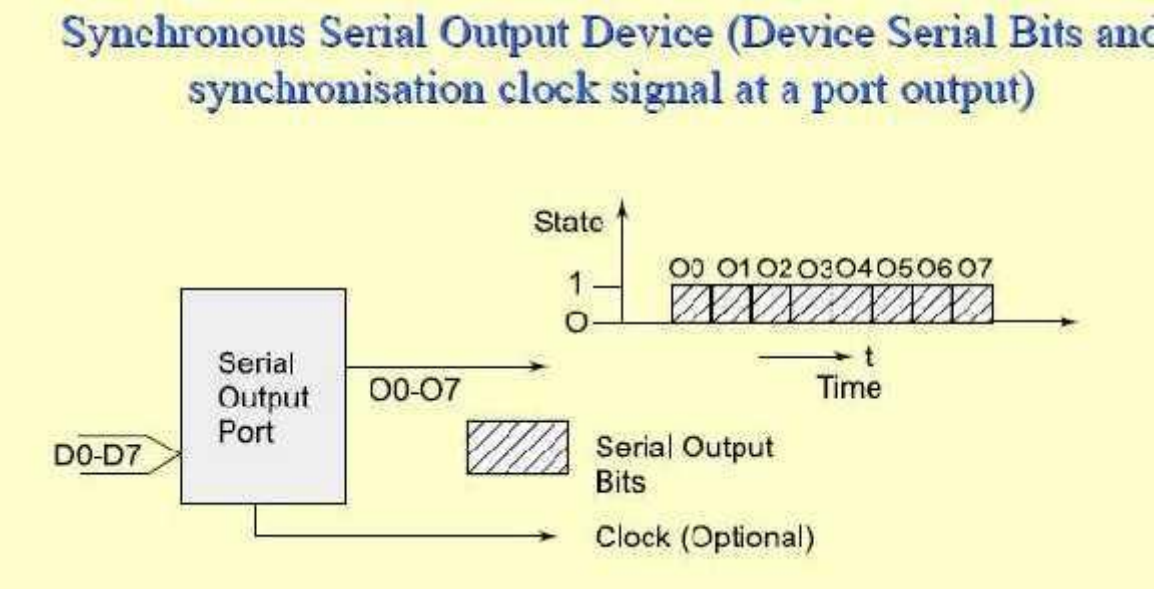
- On separate input line − When a separate SCLK input is sent, the receiver detects at the middle or+ ve edge or –ve edge of the clock pulses that whether the data-input is 1 or 0 and

saves the bits in an 8-bit shift register. The processing element at the port (peripheral) saves the byte at a port register from where the microprocessor reads the byte.

Master output slave input (MOSI) and Master input slave output (MISO)

MOSI when the SCLK is sent from the sender to the receiver and slave is forced to synchronize sent inputs from the master as per the inputs from master clock.

- MISO when the SCLK is sent to the sender (slave)from the receiver (master) and slave is forced to synchronize for sending the inputs to master as per the master clock outputs.
- Synchronous serial input is used for interprocessor transfers, audio inputs and streaming data inputs.



Example Synchronous Serial Output

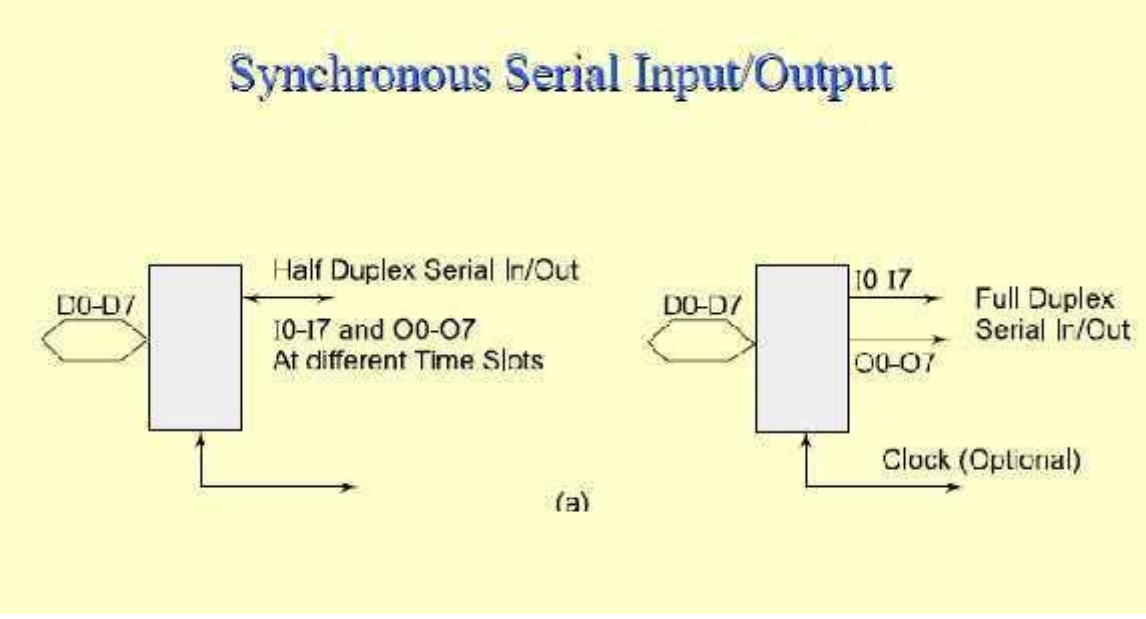
Inter-processor data transfer, multiprocessor communication, writing to CD or hard disk, audio Input/output, video Input/output,dialer output, network device output, remote TV Control, transceiver output, and serial I/O bus output or writing to flash memory using SDIO

Synchronous Serial Output

- Each bit in each byte sent in synchronization with a clock.
- Bytes sent at constant rates. If clock period= T, then data transfer rate is (1/T) bps.
- Sender either sends the clock pulses at SCLK pin or sends the serial data output and clock pulse-input through same output line with clock pulses either suitably modulate or encode the serial output bits.

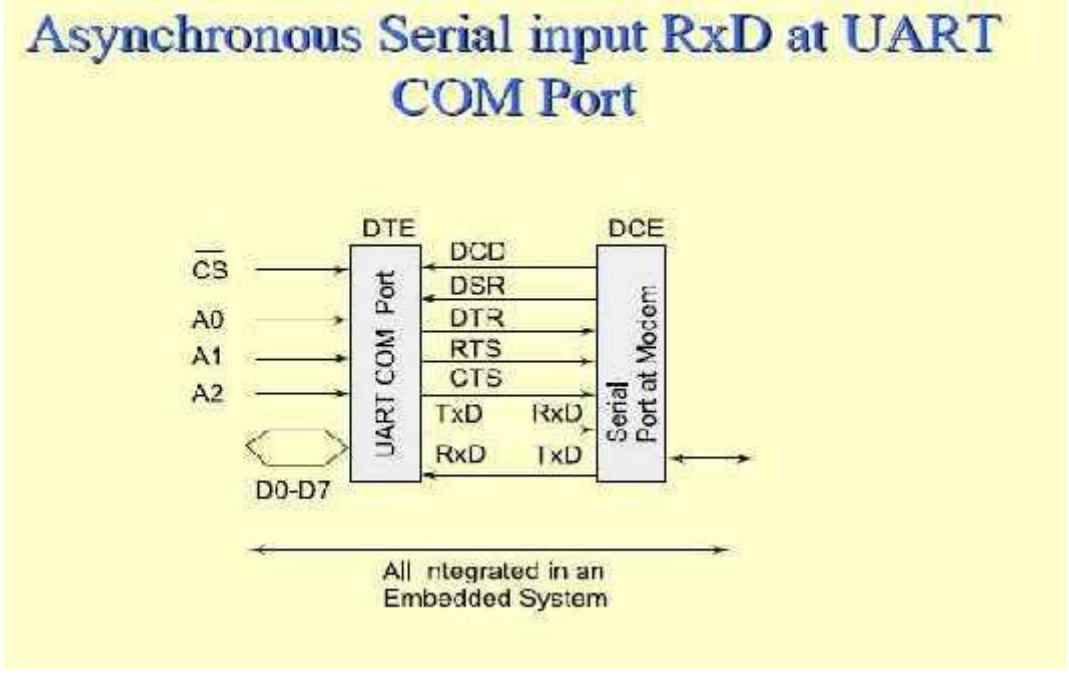
Synchronous serial output using shift register

- The processing element at the port (peripheral) sends the byte through a shift register at the port to where the microprocessor writes the byte.
- Synchronous serial output is used for inter processor transfers, audio outputs and streaming data outputs.



Synchronous Serial Input/output

- Each bit in each byte is in synchronization at input and each bit in each byte is in synchronization at output with the master clock output.
- The bytes are sent or received at constant rates. The I/Os can also be on same I/O line when input/output clock pulses either suitably modulate or encode the serial input/output, respectively. If clock period = T, then data transfer rate is (1/T)bps.
- The processing element at the port (peripheral)sends and receives the byte at a port register to or from where the microprocessor writes or reads the byte



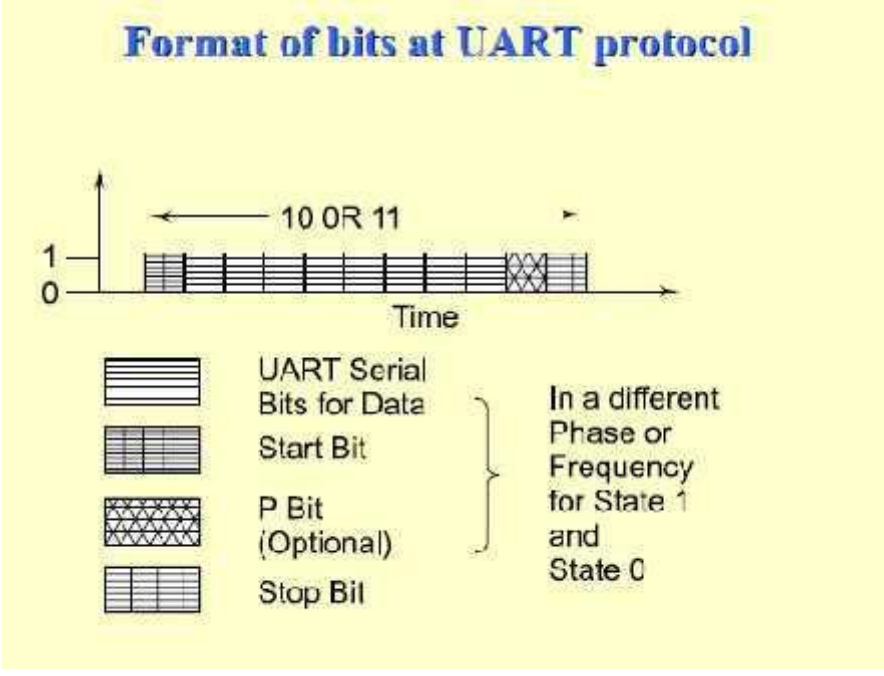
Asynchronous Serial port line RxD (receive data).

- Does not receive the clock pulses or clock information along with the bits.
- Each bit is received in each byte at fixed intervals but each received byte is not in synchronization.
- Bytes separate by the variable intervals or phase differences.
- Asynchronous serial input also called UART input if serial input is according to UART protocol

Example Serial Asynchronous Input

- Asynchronous serial input is used for keypad inputs and modem inputs in computers

- Keypad controller serial data-in, mice, keyboard controller, modem input, character send inputs on serial line [also called UART (universal receiver and transmitter) input when according to UART mode]



UART protocol serial line format

- Starting point of receiving the bits for each byte is indicated by a line transition from 1to 0 for a period = T. [T−1 called baud rate.]
- If sender‘s shift-clock period = T, then a byte at the port is received at input in period= 10.T or 11.T due to use of additional bits at start and end of each byte. Receiver detects *n* bits at the intervals of T from the middle of the start indicating bit. The *n* = 0, 1, …, 10 or 11 and finds whether the data-input is 1 or 0 and saves the bits in an 8-bit shift register.
- Processing element at the port (peripheral)saves the byte at a port register from where the microprocessor reads the byte.

Asynchronous Serial Output

- Asynchronous output serial port line TxD(transmit data).
- Each bit in each byte transmit at fixed intervals but each output byte is not in synchronization (separates by a variable interval or phase difference). Minimum separation is 1 stop bit interval TxD.
- Does not send the clock pulses along with the bits.

- Sender transmits the bytes at the minimum intervals of *n*.T. Bits receiving starts from the middle of the start indicating bit,
- n = 0, 1, …, 10 or 11 and sender sends the bits through a 10 or 11 -bit shift register.

The processing element at the port(peripheral) sends the byte at a port register to where the microprocessor is to write the byte.

- Synchronous serial output is also called UART output if serial output is according to UART protocol

Example Serial Asynchronous Output

_ Output from modem, output for printer, the output on a serial line [also called UART output when according to UART]

Half Duplex

- Half duplex means as follows: at an instant communication can only be one way (input or output) on a bi-directional line.
- An example of half-duplex mode─ telephone communication. On one telephone line, the talk can only in the half duplex way mode.

Full Duplex

- Full duplex means that at an instant,the communication can be both ways.

An example of the full duplexasynchronous mode of communicationis the communication between themodem and the computer though TxDand RxD lines or communication using SI in modes 1, 2 and 3 in 8051

Parallel Port single bit input

- Completion of a revolution of a wheel,
- Achievingpreset pressure in a boiler,
- Exceeding the upper limit of permittedweight over the pan of an electronicbalance,
- Presence of a magnetic piece in the vicinityof or within reach of a robot arm to its endpoint and Filling of a liquid up to a fixed level.

Parallel Port Output- single bit

- PWM output for a DAC, which controlsliquid level, or temperature, or pressure, orspeed or angular position of a rotating shaftor a linear displacement of an object or ad.c. motor control
- Pulses to an external circuit
- Control signal to an external circuit

Parallel Port Input- multi-bit

- ADC input from liquid level measuringsensor or temperature sensor or pressuresensor or speed sensor or d.c. motor rpmsensor
- Encoder inputs for bits for angular positionof a rotating shaft or a linear displacementof an object.

Parallel Port Output- multi-bit

- LCD controller for Multilane LCD displaymatrix unit in a cellular phone to display onthe screen the phone number, time,messages, character outputs or pictogrambit-images for display screen or e-mail orweb page
- Print controller output
- Stepper-motor coil driving bits

Parallel Port Input-Output

- PPI 8255
- Touch screen in mobile phone

Ports or DevicesCommunication and communicationprotocols

Two Modes of communication between the devices and computer system

Full Duplex – Both devices or device and computer system simultaneously communicate each other.

Half Duplex – Only one device can communicate with another at an instance

Three ways of communication betweenthe ports or devices

1. Synchronous
2. Iso-synchronous
3. Asynchronous

1. Synchronous and Iso-synchronous Communication in Serial Ports or Devices **Synchronous Communication.**

When a byte (character) or a frame (acollection of bytes) in of the data isreceived or transmitted at the constanttime intervals with uniform phasedifferences, the communication iscalled as ***synchronous***. Bits of a fullframe are sent in a prefixed maximumtime interval.

Iso-synchronous

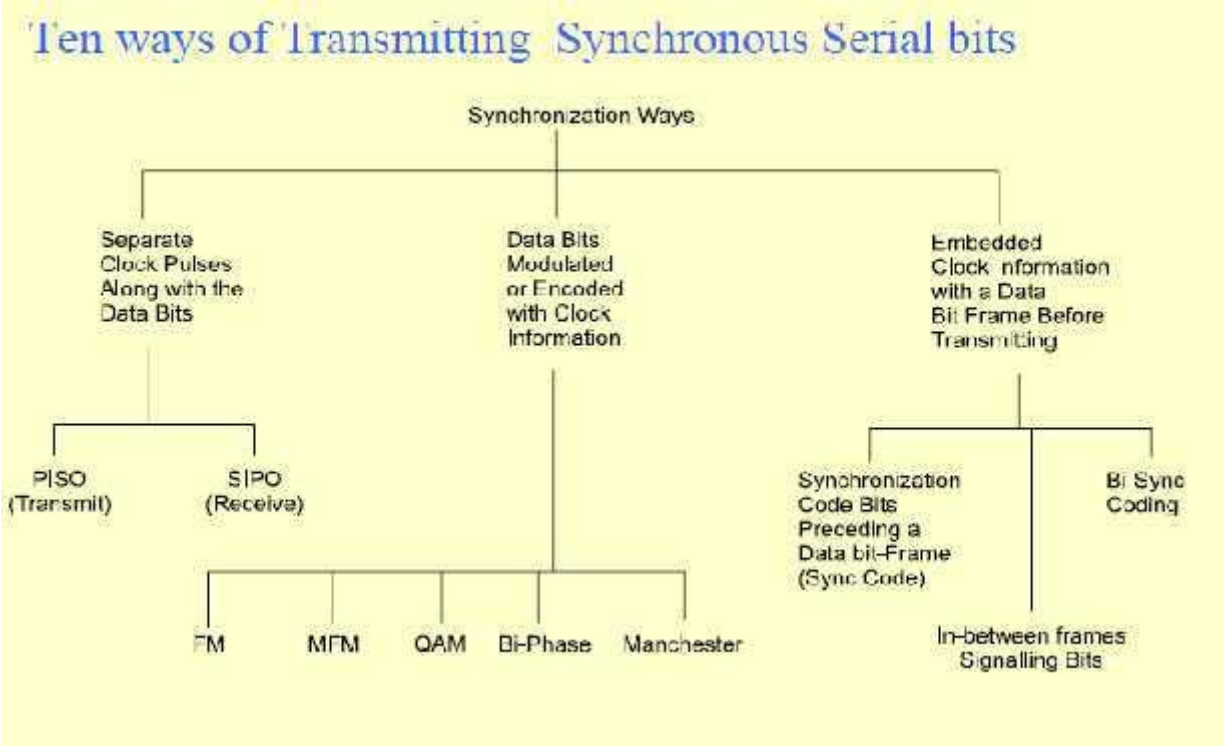
Synchronous communication special case−when bits of a full frame are sent in themaximum time interval, which can bevariable.

Synchronous Communication

Clock information is transmittedexplicitly or implicitly insynchronous communication. Thereceiver clock continuously maintainsconstant phase difference with thetransmitter clock. Bits of a data framemaintain uniform phase differenceand are sent within a fixed maximumtime interval.

Example of synchronous serial communication

- Frames sent over a LAN. Frames of data communicate with the constant time intervals between each frame remaining constant.
- Another example is the inter-processor communication in a multiprocessor system Optional Synchronous Code bits
- Optional Sync Code bits or bi-sync code bits orframe start and end signaling bits─ Duringcommunication few bits (each separated byinterval ΔT) sent as Sync code to enable the framesynchronization or frame start signaling.
- Code bits precede the data bits.
- May be inversion of code bits after each frame incertain protocols.
- Flag bits at start and end are also used in certainprotocols. Always present Synchronous device portdata bits
- Reciprocal of T is the bit per second(bps).
- Data bits─ *m* frame bits or 8 bitstransmit such that each bit is at the linefor time ΔT or, each frame is at the linefor time (*m*. T)*m* may be 8 or a large number. Itdepends on the protocolSynchronous device clock bits
- Clock bits ─ Either on a separate clockline or on data line such that the clockinformation is also embedded with thedata bits by an appropriate encoding ormodulation
- Generally not optional



First characteristics of synchronouscommunication

1. Bytes (or frames) maintain a constant phasedifference, which means they are synchronous,i.e. in synchronization. No permission ofsending either the bytes or the frames at therandom time intervals, this mode therefore doesnot provide for handshaking *during* thecommunication interval ─ *This facilitates fastdata communication at pre-fixed bps.*

Second characteristics of synchronouscommunication

2. A clock ticking at a certain rate has always tobe there for transmitting serially the bits of allthe bytes (or frames) *serially. Mostly, theclock is* ***not*** *always* ***implicit*** to thesynchronous data receiver. The transmitter*generally* transmits the clock rate information

Asynchronous Communication from SerialPorts or Devices

Asynchronous CommunicationClocks of the receiver and transmitterindependent, unsynchronized, but ofsame frequency and variable phasedifferences between bytes or bits of twodata frames, which may not be sentwithin any prefixed time interval.

Example of asynchronous communication

- UART Serial, Telephone or modemcommunication.
- RS232C communication between the UARTdevices
- Each successive byte can have variabletime-gap but have a minimum in-betweeninterval and no maximum limit for fullframe of many bytes

Two characteristics of asynchronouscommunication

1. Bytes (or frames) need not maintain a constantphase difference and are asynchronous, i.e., notin synchronization. There is permission to sendeither bytes or frames at variable timeintervals─ This*facilitates in-betweenhandshaking* between the serial transmitter portand serial receiver port

2. Though the *clock* must ticking at a certain ratealways has to be there to transmit the bits of asingle byte (or frame) serially, it is *alwaysimplicit* to the asynchronous data receiver and isindependent of the transmitter

Clock Features

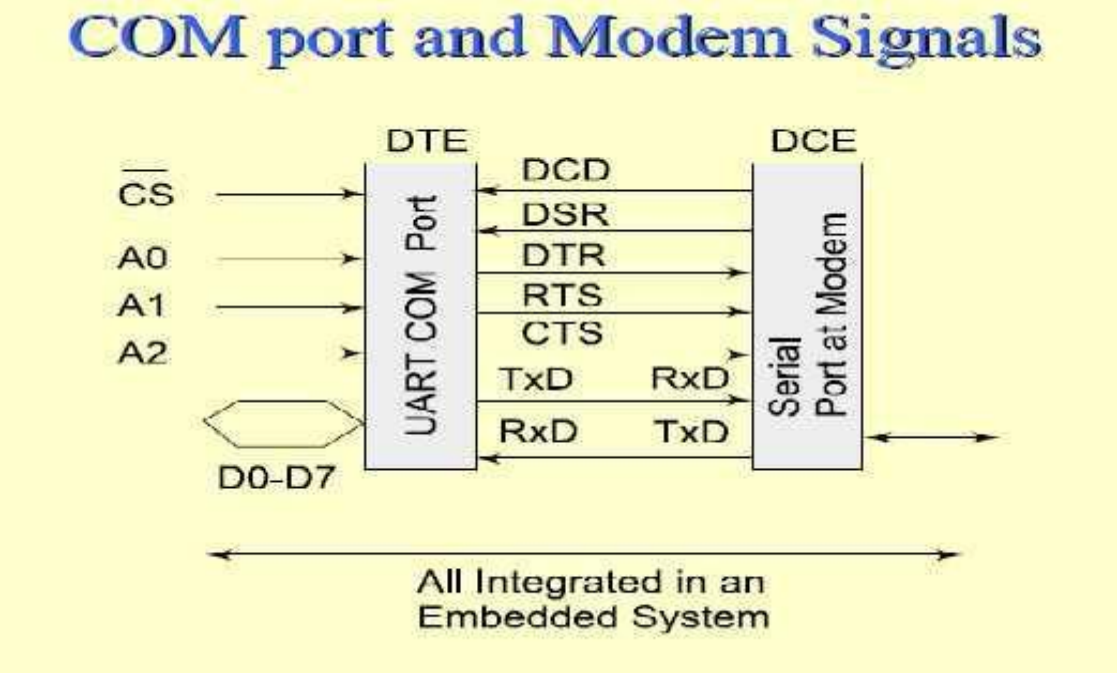
_ The transmitter *does not transmit* (neitherseparately nor by encoding using modulation)along with the serial stream of bits any *clockrate information* in the asynchronouscommunication and *receiver clock thus is notable to maintain identical frequency andconstant phase difference* with transmitter clock

Example: IBM personal computer has two COMports (communication ports)

- _ COM1 and COM2 at IO addresses 0x2F8-0xFFand 0xx38-0x3FF
- _ Handshaking signals─ RI, DCD, DSR, DTR,RTS, CTS, DTR
- _ Data Bits─ RxD and TxDExample: COM port and Modem Handshakingsignals
- _ When a modem connects, modem sends*data carrier detect* DCD signal at aninstance t0.
- _ Communicates *data set ready* (DSR)signal at an instance t1 when it receives thebytes on the line.
- _ Receiving computer (terminal) responds atan instance t2 by data terminal ready(DTR) signal.

After DTR, *request to send* (RTS) signal is sent at aninstance t3

- _ Receiving end responds by *clear to send* (CTS) signalat an instance t4. After the response CTS, the data bitsare transmitted by modem from an instance t5 to thereceiver terminal.
- _ Between two sets of bytes sent in asynchronous mode,the handshaking signals RTS and CTS can again beexchanged. This explains why the bytes do not remainsynchronized during asynchronous transmission.



3. Communication Protocols

1. Protocol

A protocol is a standard adopted,which tells the way in which the bits ofa frame must be sent from a device (orcontroller or port or processor) toanother device or system

[Even in personal communication wefollow a protocol – we say Hello! Thentalk and then say good bye!]

A protocol defines how are the framebits:

1) sent− synchronously or Isosynchronouslyor asynchronously and at what rate(s)?
2) preceded by the header bits?How the receiving device addresscommunicated so that only destineddevice activates and receives the bits?
   [Needed when several devicesaddressed though a common line(bus)]
3) How can the transmitting deviceaddress defined so that receivingdevice comes to know the sourcewhen receiving data from severalsources?
4) How the frame-length defined so thatreceiving device know the frame-sizein advance?
5) Frame-content specifications –Arethe sent frame bits specify the controlor device configuring or commend ordata?
6) Are there succeeding to frame thetrailing bits so that receiving devicecan check the errors, if any inreception before it detects end of theframe ?

A protocol may also define:

7) Frame bits minimum and maximumlength permitted per frame

8) Line supply and impedances andline-Connectors specifications

Specified protocol at an embedded systemport or communication deviceIO port bits sent after first formattedaccording to a specified protocol, whichis to be followed when communicatingwith another device through an IO portor channel

Protocols

- _ HDLC, Frame Relay, for synchronouscommunication
- _ For asynchronous transmission from a deviceport− RS232C, UART, X.25, ATM, DSL and

ADSL

- _ For networking the physical devices intelecommunication and computer networks − Ethernet and token ring protocols used in LANNetworks

Protocols in embedded network devices

- _ For Bridges and routers
- _ Internet appliances application protocolsand Web protocols ─HTTP (hyper texttransfer protocol), HTTPS (hyper texttransfer protocol Secure Socket Layer),SMTP (Simple Mail Transfer Protocol),POP3 (Post office Protocol version 3),ESMTP (Extended SMTP),

File transfer, Boot Protocols in embedded devicesnetwork

- _ TELNET (Tele network),
- _ FTP (file transfer protocol),
- _ DNS (domain network server),
- _ IMAP 4 (Internet Message ExchangeApplication Protocol) and
- _ Bootp (Bootstrap protocol).Wireless Protocols in embedded devices network
- _ Embedded wireless appliances useswireless protocols─ WLAN 802.11,802.16, Bluetooth, ZigBee, WiFi, WiMax,

TIMING ANDCOUNTING DEVICES

Timer

• Timer is a device, which counts theinput at regular interval (δT) usingclock pulses at its input.

• The counts increment on each pulseand store in a register, called countregister

• Output bits (in a count register or at theoutput pins) for the present counts.

Evaluation of Time

• The counts multiplied by the intervalδT give the time.

• The (present counts −initial counts) ×δT interval gives the time intervalbetween two instances when presentcount bits are read and initial countswere read or set.

Timer

- _ Has an input pin (or a control bit incontrol register) for resetting it for allcount bits = 0s.
- _ Has an output pin (or a status bit instatus register) for output when allcount bits = 0s after reaching themaximum value, which also meansafter timeout or overflow.

Counter

• A device, which counts the input dueto the events at irregular or regularintervals.

• The counts gives the number of inputevents or pulses since it was last read.

• Has a register to enable read of presentcounts

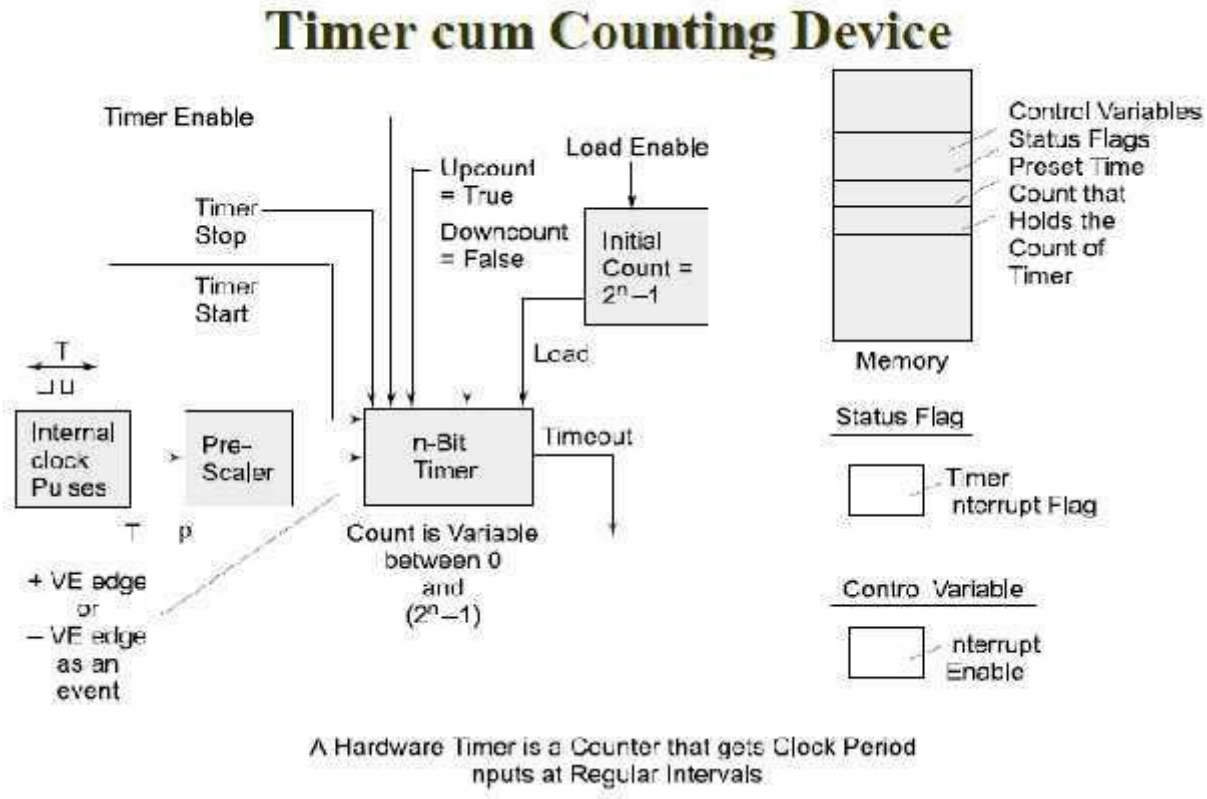
• Functions as timer when countingregular interval clock pulses

_ Has an input pin (or a control bit incontrol register) for resetting it for allcount bits = 0s.

_ Has an output pin (or a status bit instatus register) for output when allcount bits = 0s after reaching themaximum value, which also meansafter timeout or overflow.

Timer or Counter Interrupt

_ When a timer or counter becomes 0x00or 0x0000 after 0xFF or 0xFFFF(maximum value), it can generate an_interrupt‘, or an output _Time-Out‘ orset a status bit _TOV‘



Free running Counter (Blind runningCounter)

• A counting device may be a free running(blind counting) device giving overflowinterrupts at fixed intervals

• A pre-scalar for the clock input pulses to fixthe intervals

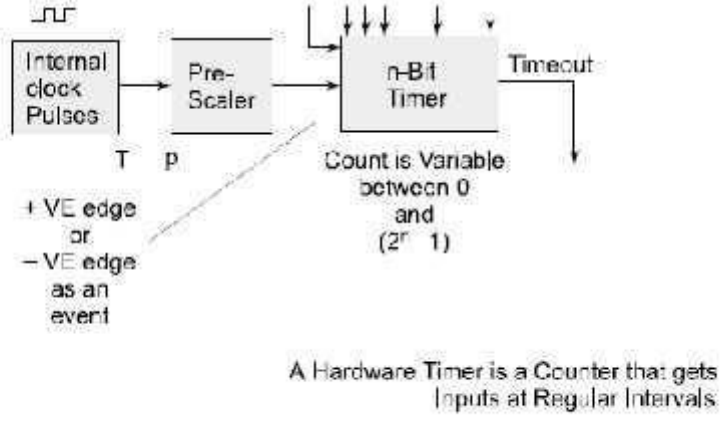
Free Running Counter

It is useful for action or initiating chain of actions,processor interrupts at the preset instances noting the instances of occurrences of theevents

_ processor interrupts for requesting theprocessor to use the capturing of counts atthe input instance

_ comparing of counts on the events for futureActions

Free running Timer cum Blind Counting Device



Free running (blind counting) device ManyApplications Based on

_ comparing the count (instance) withthe one preloaded in a compare register[an additional register for defining aninstance for an action]

_ capturing counts (instance) in anadditional register on an input event.

[An addition input pin for sensing anevent and saving the counts at theinstance of event and taking action.]

Free running (Blind Counts) input OCenablepin (or a control bit in controlregister)

• For enabling an output when all count bits atfree running count = preloaded counts in thecompare register.

• At that instance a status bit or output pin alsosets in and an interrupt _OCINT' ofprocessor can occur for event of comparisonequality.

• Generates alarm or processor interrupts atthe preset times or after **preset interval fromanother event**

Free running (Blind Counts) input capture -enable pin (or a control bit in controlregister) for Instance of Event Capture

• A register for capturing the counts onan instance of an input (0 to 1 or 1 to 0or toggling) transition

_ A status bit can also sets in andprocessor interrupt can occur for thecapture event

Free running (Blind Counts) Pre-scaling

• Prescalar can be programmed as p = 1, 2,4, 8, 16, 32, .. by programming a prescalerregister. •Prescalar divides the input pulses as perthe programmed value of *p*.

• Count interval = p × δT interval

• δT = clock pulses period, clockfrequency = δT −1

Free running (Blind Counts) Overflow

• It has an output pin (or a status bit instatus register) for output when allcount bits = 0s after reaching themaximum value, which also meansafter timeout or overflow

• Free running n-bit counter overflowsafter p × 2n × δT interval

Uses of a timer device

- _ Real Time Clock Ticks (System HeartBeats). [Real time clock is a clock, which, once the system starts, does notstop and can't be reset and its *countvalue* can't be reloaded. *Real timeendlessly flows and never returnsback*!] Real Time Clock is set for ticksusing prescaling bits (or rate set bits) inappropriate control registers.
- Initiating an event after a preset delaytime. Delay is as per *count value*loaded.
- Initiating an event (or a pair of eventsor a chain of events) after acomparison(s) with between the pre-settime(s) with counted value(s). [It issimilar to a preset alarm(s).].
- Apreset time is loaded in a CompareRegister. [It is similar to presetting analarm].
- Capturing the *count value* at the timeron an event. The information of *time*(instance of the event) is thus stored atthe *capture register*.
- Finding the time interval between twoevents. *Counts* are captured at eachevent in capture register(s) and read.The intervals are thus found out.
- Wait for a message from a queue ormailbox or semaphore for a preset timewhen using RTOS. There is aApredefined waiting period is donebefore RTOS lets a task run.

Watchdog timer.

It resets the systemafter a defined time.

- _ Baud or Bit Rate Control for serialcommunication on a line or network.Timer timeout interrupts define thetime of each baud
- _ Input pulse counting when using atimer, which is ticked by giving nonperiodicinputs instead of the clockinputs. The timer acts as a counter if, inplace of clock inputs, the inputs aregiven to the timer for each instance tobe counted.
- _ Scheduling of various tasks. A chain ofsoftware-timers interrupt and RTOSuses these interrupts to schedule thetasks.
- _ Time slicing of various tasks. Amultitasking or multi-programmedoperating system presents the illusion thatmultiple tasks or programs are runningsimultaneously by switching betweenprograms very rapidly, for example, afterevery 16.6 ms.

- _ Process known as a *context switch.*[RTOSswitches after preset time-delay from onerunning task to the next. task. Each task cantherefore run in predefined slots of time]

Time division multiplexing (TDM)

- _ Timer device used for multiplexing theinput from a number of channels.
- _ Each channel input allotted a distinctand fixed-time slot to get a TDMoutput. [For example, multipletelephone calls are the inputs and TDMdevice generates the TDM output forlaunching it into the optical fiber.

Software Timer

_ A software, which executes andincreases or decreases a count-variable(count value) on an interrupt from on asystem timer output or from on a realtimeclock interrupt.

_ The software timer also generateinterrupt on overflow of count-value oron finishing value of the countvariable.

System clock

• In a system an hardware-timing device isprogrammed to tick at constant intervals.

• At each tick there is an interrupt

• A chain of interrupts thus occur at periodicintervals.

• The interval is as per a preset*count value*

• The interrupts are called system clockinterrupts, when used to control the schedulesand timings of the system

Software timer (SWT)

• SWT is a timer based on the system clockinterrupts

• The interrupt functions as a clock input toan SWT.

• This input is common to all the SWTs thatare in the list of activated SWTs.

• Any number of SWTs can be made active ina list.

• Each SWT will set a status flag on itstimeout (*count-value* reaching 0).

• Actions are analogous to that of ahardware timer. While there is physicallimit (1, 2 or 3 or 4) for the number ofhardware timers in a system, SWTscan be limited by the number ofinterrupt vectors provided by the user.

• Certain processors (microcontrollers)also defines the interrupt vectoraddresses of 2 or 4 SWTs

SERIAL BUSCOMMUNICATION PROTOCOLS– I2C

Interconnecting number of device circuits, Assume flash memory, touch screen,ICs for measuring temperatures andICs for measuring pressures at anumber of processes in a plant.

_ ICs mutually network through acommon synchronous serial bus I2C An 'Inter Integrated Circuit' (I2C) bus,a popular bus for these circuits.

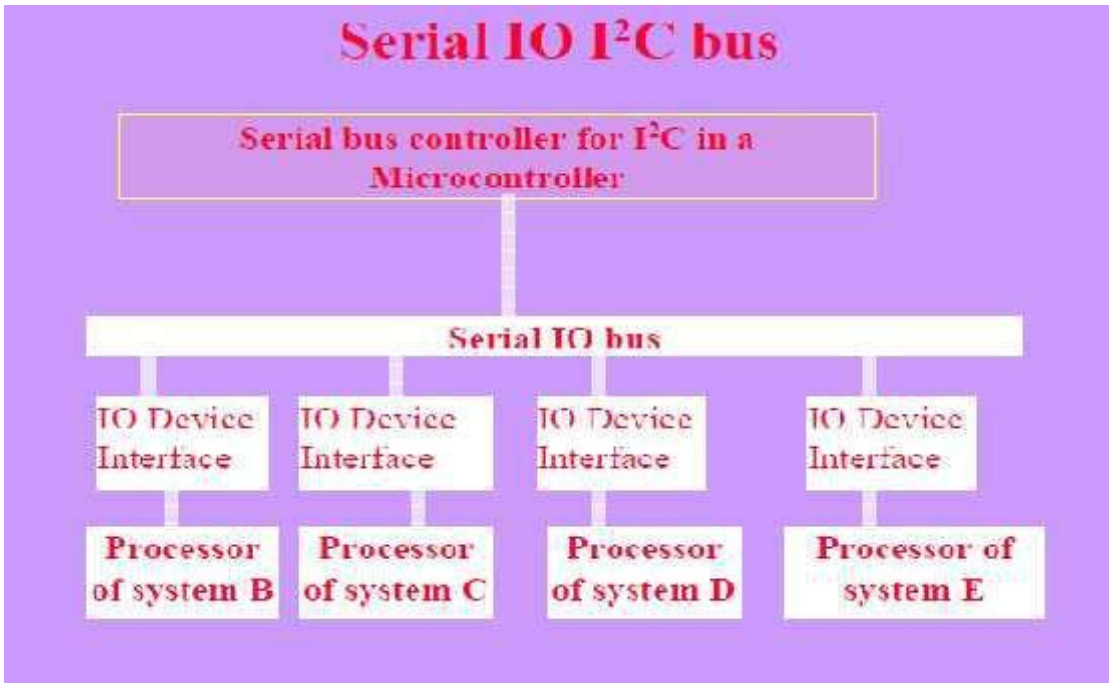
_Synchronous Serial Bus Communication fornetworking

_ Each specific I/O synchronous serial devicemay be connected to other using specificinterfaces, for example, with I/O deviceusing I2C controller

_ I2C Bus communication− use of onlysimplifies the number of connections andprovides a common way (protocol) ofconnecting different or same type of I/Odevices using synchronous serialcommunication

IO I2C Bus

_ Any device that is compatible with a I2Cbus can be added to the system(assuming an appropriate device driverprogram is available), and a I2C devicecan be integrated into any system thatuses that I2C bus.



Originally developed at PhilipsSemiconductors

Synchronous Serial Communication 400kbps up to 2 m and 100 kbps forlonger distances

Three I2C standards

1. Industrial 100 kbps I2C,
2. 100 kbps SM I2C,
3. 400 kbps I2C

I2C Bus

_ The Bus has two lines that carry itssignals— one line is for the clock andone is for bi-directional data.

_ There is a standard protocol for the I2Cbus.

Device Addresses and Master in the I2C bus

_ Each device has a 7-bit address usingwhich the data transfers take place.

_ Master can address 127 other slaves atan instance.

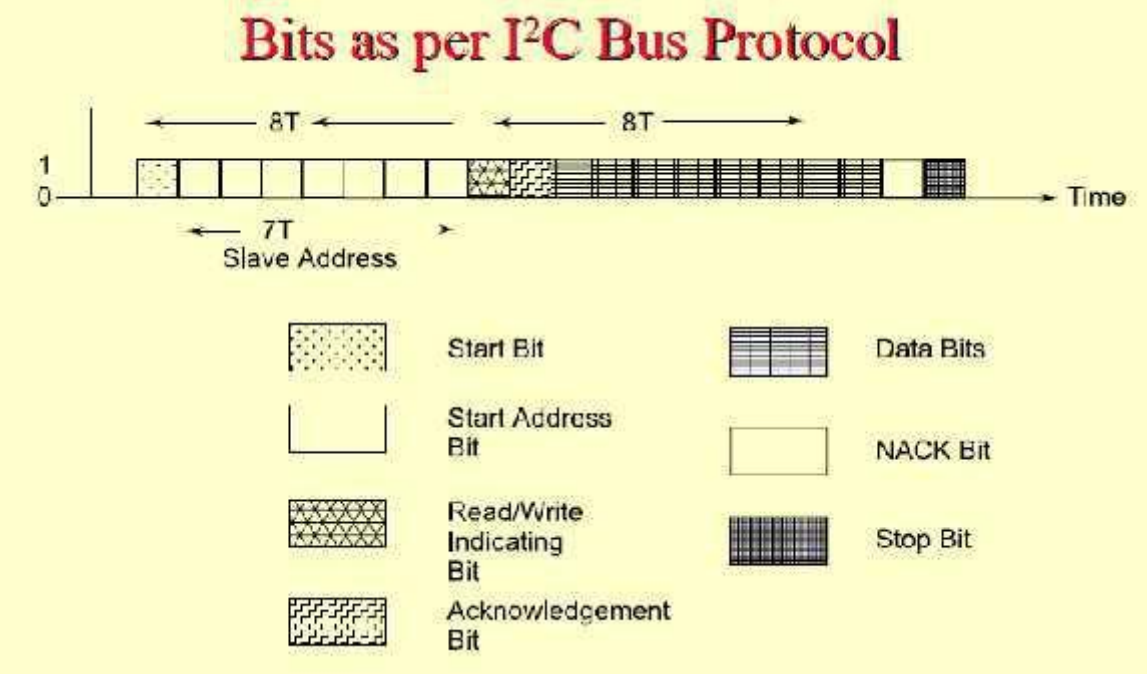
_ Master has at a processing elementfunctioning as bus controller or amicrocontroller with I2C (InterIntegrated Circuit) bus interfacecircuit.

Slaves and Masters in the I2C bus

_ Each slave can also optionally has I2C (InterIntegrated Circuit) bus controller andprocessing element.

_ Number of masters can be connected on thebus.

_ However, at an instance, master is one,which initiates a data transfer on SDA(serial data) line and which transmits theSCL (serial clock) pulses. From *master*, adata frame has fields beginning from startbit



Synchronous Serial Bus Fields and its length

_ First field of 1 bit─ Start bit similar to onein an UART

_ Second field of 7 bits─ address field. Itdefines the slave address, which is beingsent the data frame (of many bytes) by themaster

_ Third field of 1 control bit─ defineswhether a read or write cycle is in progress

_ Fourth field of 1 control bit─ defineswhether is the present data is anacknowledgment (from slave)

_ Fifth field of 8 bits─ **I2C device data byte**

_ Sixth field of 1-bit─ bit NACK (negativeacknowledgement) from the receiver. Ifactive then acknowledgment after a transferis not needed from the slave, elseacknowledgement is expected from theslave

_ Seventh field of 1 bit ─ stop bit like in anUART

Disadvantage of I2C bus

• Time taken by algorithm in thehardware that analyzes the bits throughI2C in case the slave hardware does notprovide for the hardware that supportsit.

• Certain ICs support the protocol andcertain do not.

• Open collector drivers at the masterneed a pull-up resistance of 2.2 K oneach line

SERIAL BUSCOMMUNICATION PROTOCOLS– CAN

Distributed Control Area Networkexample - a network of embeddedsystems in automobile

_ CAN-bus line usually interconnects to aCAN controller between line and host at thenode. It gives the input and gets outputbetween the physical and data link layers atthe host node.

_ The CAN controller has a BIU (businterface unit consisting of buffer anddriver), protocol controller, status-cumcontrolregisters, receiver-buffer andmessage objects. These units connect thehost node through the host interface circuit

Three standards:

1. 33 kbps CAN,
2. 110 kbps Fault Tolerant CAN,
3. 1 Mbps High Speed CAN

CAN protocol

There is a CAN controller between the CANline and the host node.

_ CAN controller ─BIU (Bus Interface Unit)consisting of a buffer and driver

_ Method for arbitration─ CSMA/AMP(Carrier Sense Multiple Access withArbitration on Message Priority basis)

Each Distributed Node Uses:

• Twisted Pair Connection up to 40 m –for bi-directional data

• Line, which pulls to Logic 1 through aresistor between the line and + 4.5V to +12V.

• Line Idle state Logic 1 (Recessivestate)

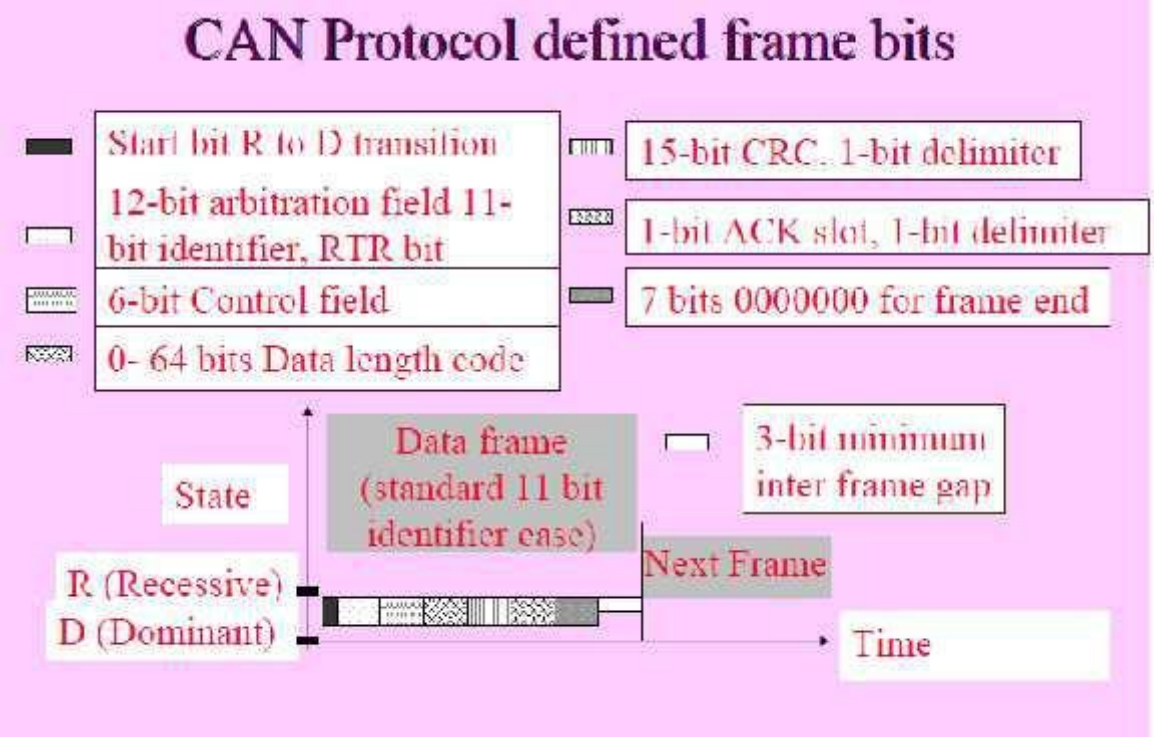
• Uses a buffer gate between an inputpin and the CAN line

• Detects Input Presence at the CAN linepulled down to dominant (active) statelogic 0 (ground ~ 0V) by a sender tothe CAN line

• Uses a current driver between theoutput pin and CAN line and pulls linedown to dominant (active) state logic 0(ground ~ 0V) when sending to theCAN lineProtocol defined start bit followed bysix fields of frame bitsData frame starts after first detecting thatdominant state is not present at the CANline with logic 1 (R state) to 0 (D statetransition) for one serial bit interval

• After start bit, six fields starting fromarbitration field and ends with seven logic0s end-field

• 3-bit minimum inter frame gap before nextstart bit (R→ D transition) occurs



Protocol defined First field in frame bits

_ First field of 12 bits ─'arbitration field.

_ 11-bit destination address and RTR bit

(Remote Transmission Request)

_ Destination device address specified in an11-bit sub-field and whether the data bytebeing sent is a data for the device or arequest to the device in 1-bit sub-field.

_ Maximum 211 devices can connect a CANcontroller in case of 11-bit address fieldstandard11-bit address standard CAN

_ Identifies the device to which data isbeing sent or request is being made.

_ When RTR bit is at '1', it means thispacket is for the device at destinationaddress. If this bit is at '0' (dominantstate) it means, this packet is a requestfor the data from the device. Protocol defined frame bits Second field

_ Second field of 6 bits─ control field.

The first bit is for the identifier‘sextension.

_ The second bit is always '1'.

_ The last 4 bits specify code for dataLength

_ Third field of 0 to 64 bits─ Its lengthdepends on the data length code in thecontrol field.

• Fourth field (third if data field has nobit present) of 16 bits─ CRC (CyclicRedundancy Check) bits.

• The receiver node uses it to detect theerrors, if any, during the transmission

• Fifth field of 2 bits─ First bit 'ACK slot'

• ACK = '1' and receiver sends back '0' in this slotwhen the receiver detects an error in the reception.

• Sender after sensing '0' in the ACK slot, generallyretransmits the data frame.

• Second bit 'ACK delimiter' bit. It signals the endof ACK field.

• If the transmitting node does not receive anyacknowledgement of data frame within a specifiedtime slot, it should retransmit.

Sixth field of 7-bits ─ end- of- theframespecification and has seven '0's

SERIAL BUSCOMMUNICATION PROTOCOLS– USB

USB Host ApplicationsConnecting

- flash memory cards,
- pen-like memory devices,
- digital camera,
- printer,
- mouse-device,
- PocketPC,
- video games,
- Scanner

Universal Serial Bus (USB)

_ Serial transmission and receptionbetween host and serial devices

_ The data transfer is of four types: (a)Controlled data transfer, (b) Bulk datatransfer, (c) Interrupt driven datatransfer, (d) Iso-synchronous transfer

_ A bus between the host system andinterconnected number of peripheraldevices

USB Protocol Features

_ Maximum 127 devices can connect ahost.

_ Three standards: USB 1.1 (a low speed1.5 Mbps 3 meter channel along with ahigh speed 12 Mbps 25 meter channel),USB 2.0 (high speed 480 Mbps 25meter channel), and wireless USB(high speed 480 Mbps 3 m)

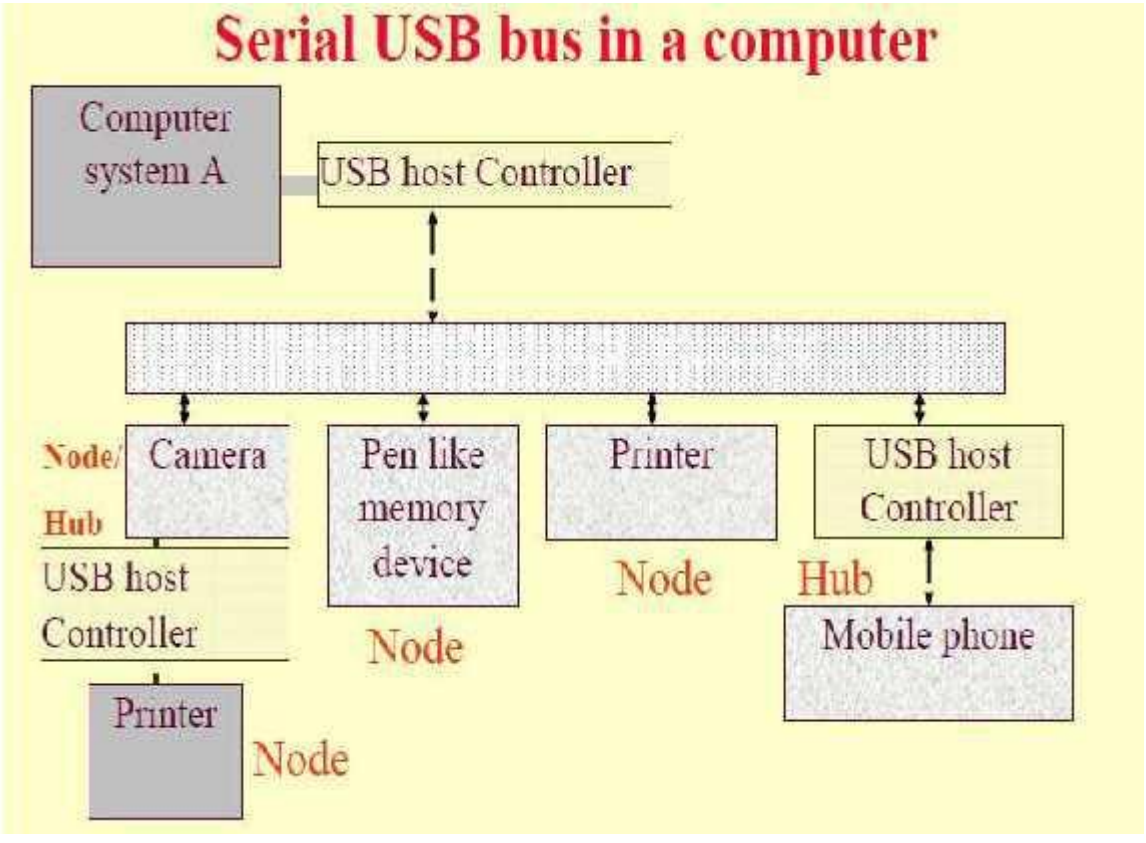
Host connection to the devices or nodes

_ Using USB port driving software andhost controller,

_ Host computer or system has a hostcontroller,which connects to a roothub.

_ A hub is one that connects to othernodes or hubs.

_ A tree- like topology



USB Device features

_ Can be hot plugged (attached), configuredand used, reset, reconfigured and used

_ Bandwidth sharing with other devices: Hostschedules the sharing of bandwidth amongthe attached devices at an instance.

_ Can be detached (while others are inoperation) and reattached.

_ Attaching and detaching USB device orhost without rebooting

USB device descriptor

_ Has data structure hierarchy asfollows:

_ It has device descriptor at the root,which has number of configurationdescriptors, which has number ofinterface descriptor and which hasnumber of end point descriptor.

Powering USB device

_ A device can be either bus-powered orself- powered.

_ In addition, there is a powermanagement by software at the host forUSB ports

USB protocol

_ USB bus cable has four wires, one for+5V, two for twisted pairs and one forground.

_ Termination impedances at each end asper the device-speed.

_ Electromagnetic Interference (EMI)-shielded cable for the 15 Mbps USBdevices.

_ Serial signals NRZI (Non Return toZero (NRZI)

_ The synchronization clock encoded byinserting synchronous code (SYNC)field before each USB packet

_ Receiver synchronizes its bits recoveryclock continuously

USB Protocol

• A polled bus

• Host controller regularly polls the presenceof a device as scheduled by the software.

• It sends a token packet.

• The token consists of fields for type,direction, USB device address and deviceend-point number.

• The device does the handshaking through ahandshake packet, indicating successful orunsuccessful transmission.

• A CRC field in a data packet permitserror detection

USB supported three types of pipes

1. 'Stream' with no USB- defined protocol. Itis used when the connection is alreadyestablished and the data flow starts
2. 'Default Control' for providing access.
3. 'Message' for the control functions for of thedevice.

• Host configures each pipe with the databandwidth to be used, transfer service typeand buffer sizes.

PARALLEL BUSDEVICE PROTOCOLS – PCI Bus

_ Parallel bus enables a host computer orsystem to communicate simultaneously 32-bit or 64-bit with other devices orsystems, for example, to a networkinterface card (NIC) or graphic card

Computer system PCI

• When the I/O devices in the distributedembedded subsystems are networked allcan communicate through a commonparallel bus.

• PCI connects at high speed to othersubsystems having a range of I/O devicesat very short distances (<25 cm) using aparallel bus without having to implementa specific interface for each I/O device.

PCI bus Applications

Connects

_ display monitor,

_ printer,

_ character devices,

_ network subsystems,

_ video card,

_ modem card,

_ hard disk controller,

PCI busconnects

_ thin client,

_ digital video capture card,

_ streaming displays,

_ 10/100 Base T card,

_ Card with 16 MB Flash ROM with a routergateway for a LAN

and _ Card using DEC 21040 PCI Ethernet LANcontroller.

• When the I/O devices in the distributedembedded subsystems are networked, allcan communicate through a commonparallel bus.

• PCI connects at high speed to othersubsystems having a range of I/O devicesat very short distances (<25 cm) using aparallel bus without having to implementa specific interface for each I/O device.

PCI Bus Feature

_ 32- bit data bus extendible to 64 bits.

_ PCI protocol specifies the ways ofinteraction between the differentcomponents of a computer. _ A specification version 2.1─synchronous/asynchronous throughputis up to 132/ 528 MB/s [33M × 4/ 66M× 8 Byte/s], operates on 3.3V to 5Vsignals.

_ PCI driver can access the hardwareautomatically as well as by theprogrammer assigned addresses.

_ Automatically detects the interfacingsystems and assigns new addresses

_ Thus, simplified addition and deletion(attachment and detachment) of thesystem peripherals.

FIFO in PCI device/card

_ Each device may use a FIFO controllerwith a FIFO buffer for maximumthroughput. Identification Numbers

_ A device identifies its address space bythree identification numbers, (i) I/Oport (ii) Memory locations and (iii)Configuration registers of total 256Bwith a four 4-byte unique ID. Each PCIdevice has address space allocation of256 bytes to access it by the host Computer

PCI device identification

_ A sixteen16-bit register in a PCI deviceidentifies this number to let that deviceauto- detect it.

_ Another sixteen16-bit registeridentifies a device ID number. Thesetwo numbers let allow the device tocarry out its auto-detection by its hostcomputer.

Peripheral Component Interconnect (PCI) Bus

_ Independent from the IBMarchitecture.

_ Number of embedded devices in acomputer system use PCI

_ Three standards for the devicesinterfacing with the PC _

PCI 32bit/33 MHz, and 64bit/66 MHz

_ PCI Extended (PCI/X) 64 bit/100 MHz ,

_ Compact PCI (cPCI) Bus

Two super speed versions

_ PCI Super V2.3 264/528 MBps 3.3V (on64- bit bus), and 132/264 (on 32-bit bus)and

_ PCI-X Super V1.01a for 800MBps 64- bitbus 3.3Volt.

PCI bridge

_ PCI bus interface switches a processorcommunication with the memory bus to PCIbus. _

In most systems, the processor has a singledata bus that connects to a switch module

_ Some processors integrate the switchmodule onto the same integrated circuit asthe processor to reduce the number of chipsrequired to build a system and thus the system cost.

_ Communicates with the memorythrough a *memory bus* (a set ofaddress, control and data buses), adedicated set of wires that transfer databetween these two systems. _ A separate *I/O bus* connects the PCIswitch to the I/O devices.

Advantage of Separate memory and I/Obuses

_ I/O system generally designed formaximum flexibility, to allow as manydifferent I/O devices as possible tointerface to the computer

_ Memory bus is designed to provide themaximum-possible bandwidth betweenthe processor and the memory system.

PCI-X (PCI extended)

• 133 MBps to as much as 1 GBps

• Backward compatible with existing

PCI cards

• Used in high bandwidth devices(Fiber Channel, and processors thatare part of a cluster and GigabitEthernet)

• Maximum 264 MBpsthroughput, uses 8,16, 32, or 64 bit transfers

• 6U cards contain additional pins for userdefined I/Os

• Live insertion support (Hot-Swap),

• Supports two independent buses on theback plane (on different connectors)

• Supports Ethernet, Infiniband, and StarFabric support (Switched fabric basedsystems) **Compact PCI (cPCI)**

Each PCI device on Bus

_ Perform a specific function,

_ May contain a processor and software toperform a specific function.

_ Each device has the specific memoryaddress-range, specific interrupt-vectors(pre-assigned or auto configured) and thedevice I/O port addresses.

_ A bus of appropriate specifications andprotocol interfaces these to the hostcomputer system or compute

Configuration address space

_ Unique feature of PCI bus uniquefeature is its configuration addressspace.

PCI controller Features

• Accesses one device at a time

• All the devices within host device orsystem can share the I/O port andmemory addresses, but cannot sharethe configuration registers

• Device cannot modify otherconfiguration registers but can accessother device resources or share thework or assist the other device

• If there are reasons for doing it so, aPCI driver can change the default bootup assignments on configurationtransactions.

PCI Device Initialization

A device can initialize at booting time

• Avoids any address collision

• Device on boot up disables its interruptand closes its door to its address spaceexcept to the configuration registersspace

PCI BIOS (Basic Input-Output System)

Performs the configuration transactionsand then, memory and address spacesautomatically map to the address spacein the device hosting system

UNIT III PROGRAMMING CONCEPTS AND EMBEDDED PROGRAMMING IN C, C++ 9

Programming in Assembly and HLL

• Processor and memory-sensitiveinstructions: Program codes maybe written in assembly
• Most of codes: Written in a highlevel language (HLL), _C', _C++'or Java

Assembly Language Programming

Advantage

- Assembly codes sensitive to the processor,memory, ports and devices
- hardware Gives a precise control of the processorinternal devices
- Enables full use of processor specific featuresin its instruction set and its addressing modes
- Machine codes are *compact*, processor andmemory sensitive
- System needs a smaller memory.
- Memory needed does not depend on theprogrammer data type selection and ruledeclarations
- Not the compiler specific and libraryfunctionsspecific
- Device driver codes may need only a fewassembly instructions.
- Bottom-up-design approach

Advantage of using high levellanguage (HLL) for Programming

Short Development Cycle

• Code reusability─ A function or routinecan be repeatedly used in a program
• Standard library functions─ Forexamples, the mathematical functionsand delay ( ), wait ( ), sleep ( ) functions
• Use of the modular building blocks
• Sub-modules are designed first forspecific and distinct set of actions,then the modules and finallyintegration into complete design.
• First code the basic functionalmodules and then build a biggermodule and then integrate into thefinal system
• First design of main program (blueprint),then its modules and finallythe sub-modules are designed forspecific and distinct set of actions.
• Top-down design Most favouredprogram design approach

Use of Data Type and Declarations

• Examples, *char, int, unsigned short,long, float, double, Boolean*.
• Each data type provides anabstraction of the (i) methods to use,manipulate and represent, and (ii) setof permissible operations.

Use of Type Checking

• Type checking during compilationmakes the program less prone toerrors.
• Example─ type checking on a chardata type variable (a character) doesnot permit subtraction, multiplicationand division.

Use of Control Structures, loops andConditions

• Control Structures and loops
• Examples─ *while*, *do-while*, *break*and *for*
• Conditional Statements examples
• *if, if- else, else - if* and *switch - case*)

• Makes tasks simple for the programflowDesign

Use of Data Structures

_ Data structure

- A way of organizinglarge amounts of data.

_ A data elements‘ collection

_ Data element in a structure identifiedand accessed with the help of a fewpointers and/or indices and/orfunctions.

Standard Data structure

• Queue
• Stack
• Array – one dimensional as a vector
• Multidimensional
• List
• Tree

Use of Objects

_ Objects bind the data fields andmethods to manipulate those fields

_ Objects reusability

_ Provide inheritance, methodoverloading, overriding and interfacing _ Many other features for ease inprogramming

Advantage of using C for Programming

C

•Procedure oriented language (Noobjects)

• Provision of inserting the assemblylanguage codes in between (called inlineassembly) to obtain a directhardware control.

• A large program in ‗C‘ splits into thedeclarations for variables, functions anddata structure, simpler functional blocksand statements.

In-line assembly codes of C functions

• Processor and memory sensitivepart of the program within the inlineassembly, and the complexpart in the HLL codes.

• Example function ouportb (q, p)

• Example─ *Mov al, p; out q, al*

C Program Elements

Preprocessor include Directive

_ Header, configuration and otheravailable source files are madethe part of an embedded systemprogram source file by thisdirective

Examples of Preprocessor includeDirectives

include "*VxWorks.h*" /* IncludeVxWorks functions*/
include "*semLib.h*" /* IncludeSemaphore functions Library */
include "*taskLib.h*" /* Includemultitasking functions Library */
include "*sysLib.c*" /* Include system libraryfor system functions */
include "*netDrvConfig.*txt" /* Include a textfile that provides the 'Network DriverConfiguration'. */
include "*prctlHandlers.c*" /* Include file forthe codes for handling and actions as perthe protocols used for driving streams tothe network. */

Preprocessor Directive for theDefinitions

• Global Variables ─ *# definevolatile booleanIntrEnable*
• Constants ─ *# define false 0*

• Strings─ # *define welcomemsg*"Welcome To ABC Telecom"

Preprocessor Macros

• Macro - A named collection of codes that isdefined in a program as preprocessor directive.
• Differs from a function in the sense that oncea macro is defined by a name, the compilerputs the corresponding codes at the macro atevery place where that macro-name appears.re used for short codesonly.

Difference between Macro and Function

• The codes for a function compiledonce only
• On calling that function, theprocessor has to save the context,and on return restore the context.
• Macros are used for short codesonly.
• When a function call is used instead ofmacro, the overheads (context savingand return) will take a time, Toverheads that is the same order of magnitude asthe time, Texec for execution of shortcodes within a function.
• Use the function when the Toverheads<<Texec and macro when Toverheads ~= or > Texec.

Use of Modifiers

- auto
- unsigned
- static
- const
- register
- interrupt
- extern
- volatile
- volatile static

Use of infinite loops

_ Infinite loops- Never desired inusual programming. Why? Theprogram will never end and neverexit or proceed further to the codesafter the loop.
_ Infinite loop is a feature inembedded system programming!
Example:
A telephone is never switching off.
The system software in thetelephone has to be always in awaiting loop that finds the ring onthe line. An exit from the loop willmake the system hardwareredundant.

```
# define false 0
# define true 1
void main (void) {
/* Call RTOS run here
*/ rtos.run ( );
/* Infinite while loops follows in eachtask. So never there is return from theRTOS. */
}
void task1 (....) {
/* Declarations
*/. while (true) {
/* Run Codes that repeatedly execute */
/* Run Codes that execute on an event*/
if (flag1) {....;}; flag1 =0;
/* Codes that execute for message to thekernel */
message1 ( ); } }
```

Use of typedef
_ Example─ A compiler version may notprocess the declaration as an unsigned byte _ The 'unsigned character' can then be used asa data type.
_ Declared as follows: typedef unsignedcharacter portAdata
_ Used as follows: #define PbyteportAdata0xF1

Use of Pointers

Pointers are powerful tools whenused correctly and according tocertain basic principles.

 # define COM ((structsio near*) 0x2F8);

This statement with a single masterstroke assigns the addresses to all 8variables

Byte at the sio Addresses

0x2F8: Byte at RBR/THR /DLATCH-LByte
0x2F9: Byte at DLATCH-HByte
0x2FA: Byte at IER; 0x2FB: Byte at LCR;
0x2FC: Byte at MCR;
0x2FD: Byte at LSR; 0x2FE: Byte at MSR
0x2FF: Byte Dummy Character

Example

Free the memory spaces allotted to a datastructure.
#define NULL (void) 0x0000*
• Now statement *& COM ((structsionear*) = NULL;*assigns the COM to Null and make freethe memory between 0x2F8 and 0x2FFfor other uses.

Data structure
• Example─ structure *sio*
• Eight characters─ Seven for thebytes in BR/THR/DLATCHLByte,IER, IIR, LCR, MCR, LSR, MSRregisters of serial line device andone dummy variable**re consisting of 8 charactervariables structure for the COM port 2 inthe UART serial line device at an IBMPC.**

Example of Data structure declaration

• Assume structured variable COM at theaddresses beginning 0x2F8.
 # define COM ((structsio near*) 0x2F8)
• COM is at 8 addresses 0x2F8-0x2FF andis a structure consisting of 8 charactervariables structure for the COM port 2 inthe UART serial line device at an IBMPC.
 # define COM1 ((structsio near*) 0x3F8);
It will give another structured variableCOM1 at addresses beginning 0x3F8using the data structure declared earlieras *sio*

Use of functions

(i) Passing the Values (elements):
The values are copied into thearguments of the functions. Whenthe function is executed in thisway, it does not change a variable'svalue at the function, which callsnew function.
(ii) Passing the References
When an argument value to afunction passes through a pointer,the called function can change thisvalue. On returning from thisfunction, the new value may beavailable in the calling program oranother function called by thisfunction.

Use of Reentrant Function

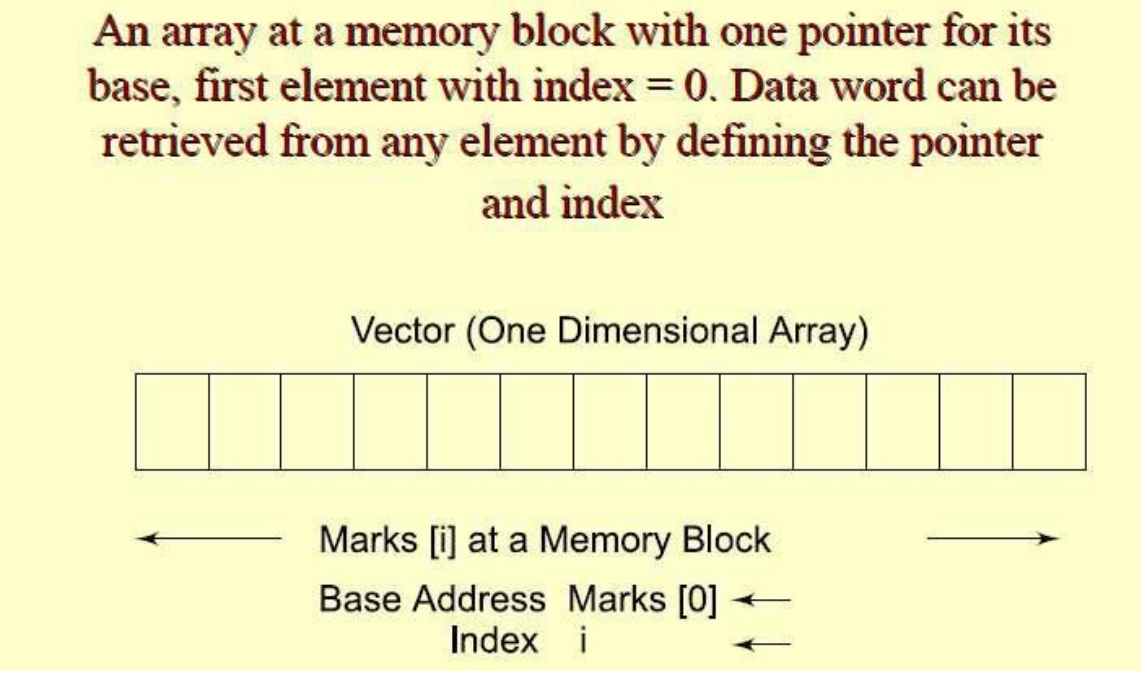
• Reentrant function- A functionusable by the several tasks androutines synchronously (at thesame time). This is because all thevalues of its argument areretrievable from the stack.

Three conditions for a function calledas *reentrant function*

1. All the arguments pass the valuesand none of the argument is apointer (address) whenever acalling function calls that function.
2. When an operation is not atomic, thatfunction should not operate on anyvariable, which is declared outside thefunction or which an interrupt serviceroutine uses or which is a globalvariable

but passed by reference andnot passed by value as an argumentinto the function. [The value of such avariable or variables, which is notlocal, does not save on the stack whenthere is call to another program.]
3. That function does not call any otherfunction that is not itself Reentrant.

Data Structures: Arrays
• Array: A structure with a series ofdata items sequentially placed inmemory
(i) Each element accessible by anidentifier name (which points tothe array) and an index, *i* (whichdefine offset from the firstelement)
(ii) *i*starts from 0 and is +ve integer



One dimensional array (vector)
Example 1:
unsignedint*salary* [*11*];
salary[0] – 1st month salary.
salary[11] – 12th month salary
Each integer is of 32-bit (4 bytes);
*salary*assigned 48 bytes addressspace

Example 2: sio*COM* [1];
COM [0]– COM1 port data record with structure equivalent to *sio*
COM [1]– COM2 port data record with structure equivalent to *sio*
COM assigned 2*8 characters = 16 bytes address space

Two dimensional array
Example 3:
unsignedint*salary* [11, 9];
salary[3, 5]– 4th month 6th year salary
salary[11, 4] – 12th month 5th yearsalary
*salary*assigned 12*10*4 = 480 bytesaddress space

Multi-dimensional array
Example 4:
char*pixel* [*143,175, 23*];
pixel [0, 2, 5] – 1st horizontal line index *x*,3rd vertical line index *y*, 6th color *c.pixel* assigned
144*176*24 = 608256bytes address space in a coloredpicture of resolution 144x 176 and 24colors.

Programming using functions andfunction queues _
Use of multiple function calls in the main ( )
_ Use of multiple function calls in cyclic order
_ Use of pointer to a function
_ Use of function queues and
_ Use of the queues of the function pointers built bythe ISRs.
It reduces significantly the ISR latencyperiods. Each device ISR is therefore able toexecute within its stipulated deadline

1.Multiple function calls
2. Multiple function calls in cyclic order
Use
• One of the most commonmethods is the use of multiplefunction-calls in a cyclic order inan infinite loop of the *main* ( ).
3. Use of function pointers
* sign when placed before thefunction name then it refers to allthe compiled form of thestatements in the memory that arespecified within the curly braceswhen declaring the function.
• A returning data type specification (forexample, void) followed by'*(*functionName) (functionArguments)*'calls the statements of the*functionName*using the*functionArguments*, and on a return, itreturns the specified data object. Wecan thus use the function pointer forinvoking a call to the function.
4. Queue of Function-pointers Application
_ Makes possible the designing ofISRs with short codes and byrunning the functions of the ISRsat later stageso all pending ISRsfinishes
Multiple ISRs insertion of Function pointers into a Queue
• The ISRs insert the function pointers
• The pointed functions in the queue execute at later stages by deleting from the queue
• These queued functions execute after the service to all pending ISRs finishes
Priority Function Queue of Multiple ISRs
• When there are multiple ISRs, a high priority interrupt service routine is executed first and the lowest priority.
• The ISRs insert the function pointers into a priority queue of function pointers[ISR can now be designed short enough sothat other source don‘t miss a deadline forservice]

CONCEPTS ANDEMBEDDED PROGRAMMING INC, C++

Multitasking

Function *main* with a waiting loop

main ( ) passes the control to an RTOS
Each task controlled by RTOS and
Each task will also have the codes in aninfinite loop A
waiting task is passed a signal by theRTOS to start.

main ( ) calling RTOS

```
 # define false 0
 # define true 1
 /*************************************************************************/
void main (void) {
 /* Call RTOS run here */
```

Infinite loop in main ( )

```
while (1) {rtos.run ( );
 /* Infinite while loops follows in each task.So never there is return from the RTOS. */
 }
 }
 /*************************************************************************/
```

Task 1

```
void task1 (....) {
 /* Declarations */
.
while (true) {
 /* Codes that repeatedly execute */
.
 /* Codes that execute on an event */
if (flag1) {....;}; flag1 =0;
 /* Codes that execute for message to the kernel */
 message1 ( );
 } }
 /*********************************************/
```

Task2 ( )

```
void task2 (....) {
 /* Declarations */
.
while (true) {
 /* Codes that repeatedly execute */
.
 /* Codes that execute on an event */
if (flag2) {....;}; flag2 =0;
 /* Codes that execute for message to the kernel */
 message2 ( );
 } }
 /*********************************************/
```

TaskN_1 ( )

```
void taskN_1 (....) {
 /* Declarations */
.
while (true) {
 /* Codes that repeatedly execute */
.
 /* Codes that execute on an event */
if (flagN_1) {....;}; flagN_1 =0;
 /* Codes that execute for message to the kernel */
```

```
 messageN_1 ( );
 } }
 /*********************************************/
```

TaskN

```
voidtaskN (....) {
 /* Declarations */
.
while (true) {
 /* Codes that repeatedly execute */
.
 /* Codes that execute on an event */
if (flagN) {....;}; flagN =0;
 /* Codes that execute for message to the kernel */
messageN ( );
 } }
 /*********************************************/
```

2. Polling for events and messages

_ A Programming method is to facilitate execution of one of the multiple possible function calls and the function executes after polling

_ Polling example is polling for a screen state (or Window menu) *j* and for a message *m* from an

Mobile phone

_ Assume that screen state j is between 0 and K,among 0, 1, 2, ..or K – 1 possible states.(set ofmenus).

_ An interrupt is triggered from a touch screen GUIand an ISR posts an event-message *m* = 0, 1, 2, …,or N – 1 as per the selected the menu choice 0, 1,2, …, N – 1 when there are N menu- choices for amobile phone user to select from a screen in statej.

Polling for a menu selection from screen state

```
voidpoll_menuK {/* Code for polling forchoice from menu m for screen state K*/
 }
 }
 /*********************************/
```

Object OrientedLanguage and C++

Object-oriented language features

_ defining of the object or set of objects,which are common to similar objectswithin a program and between the manyprograms,

_ defining the methods that manipulate theobjects without modifying theirdefinitions, and

_ Creation of multiple instances of thedefined object or set of objects or newobjects

Object-oriented language

_ Inheritance

_ overloading of functions

_ overriding of functions

_ Data encapsulation, and
_ Design of reusable components

Object Characteristics

1. An i*dentity* (a reference to amemory block that holds its stateand behavior).
2. A *state* (its data, property, fieldsand attributes).
3. A *behavior*(method or methodsthat can manipulate the *state* ofthe object).

Procedure oriented language

Procedure oriented language Alarge program in _C' splits intothe simpler functional blocks andstatements. _C' is calledprocedure oriented language.

Object Oriented Language Characteristics

• A large program in objected orientedlanguage C++ or Java, splits into thelogical groups (also known as ***classes***).
• Each class defines the data andfunctions (methods) of using data.
• Each class can inherit another classelement.
• A set of these groups (classes) then givesan application program of the EmbeddedSystem
• Each group has internal user-level fieldsfor data and has methods of processingthat data at these fields
• Each group can then create many objectsby copying the group and making itfunctional.
• Each object is functional. Each objectcan interact with other objects toprocess the user's data.
• The language provides for formationof classes by the definition of a groupof objects having similar attributesand common behavior. A class*creates the* ***objects.*** **An object is aninstance of a class.**

Embedded Programming in C++

• C++ is an object oriented Program(OOP) language, which inaddition, supports the procedureoriented codes of C.
• Program coding in C++ codesprovides the advantage of objectedoriented programming as well asthe advantage of C and in-lineassembly.

C++

_ *struct*that binds all the member functionstogether in C. But a C++ *class* has objectfeatures. It can be extended and childclasses can be derived from it. A number ofchild classes can be derived from a commonclass. This feature is called polymorphism.

A class can be declared as public or private.The data and methods access is restrictedwhen a class is declared private. *Struct*doesnot have these features.

_ A class binds all the member functions togetherfor creating objects. The objects will have memoryallocation as well as default assignments to itsvariables that are not declared *static.*

_ A class can derive (inherit) from another classalso. Creating a *child* class from RTCSWT as a*parent* class creates a new application of theRTCSWT.

_ Methods (C functions) can have same name in theinherited class. This is called *method overloading*

_ Methods can have the same name as well asthe same number and type of arguments in the inherited class. This is called *methodoverriding*. These are the two significantfeatures that are extremely useful in a largeprogram.

_ Operators in C++ can be overloaded like inmethod overloading.
_ For example, operators ++ and! areoverloaded to perform a set of operations.

Some disadvantages

• Lengthier Code when using Template,Multiple Inheritance (Deriving a class frommany parents), Exceptional handling, Virtualbase classes and classes for IO Streams.

Ways to overcome the disadvantages

1) Declare private as many classes aspossible. It helps in optimising thegenerated codes.
2) Use *char*, *int*and *boolean*(scalar datatypes) in place of the objects (referencedata types) as arguments and use localvariables as much as feasible.
3) Recover memory already used once bychanging the reference to an object toNULL.
4) A ***special compiler for an embeddedsystem*** can facilitate the disabling ofspecific features provided in C++.**Embedded C++** is a version of C++ thatprovides for a selective disabling of theabove features
5) Use Embedded C++: It provides for lessruntime overhead and less runtimelibrary. The solutions for the libraryfunctions are available and ported in Cdirectly.
6) The IO stream library functions in anembedded C++ compiler are alsoreentrant.
7) Using embedded C++ compilers or thespecial compilers make the C++ morepowerful coding language than C forembedded systems due to the OOP featuresof software re-usability, extendibility,polymorphism, function overriding andoverloading along portability of C codes andin-line assembly codes.

UNIT IV REAL TIME OPERATING SYSTEMS – PART - 1 9

Process Concepts

- A process consists of executable program (codes), *state* of which is controlled by OS, the *state* during running of a process represented by process-status (running, blocked, or finished), process structure—its data, objects and resources, and process control block (PCB).
- Runs when it is scheduled to run by the OS (kernel)
- OS gives the control of the CPU on a process‘s request (system call).
- Runs by executing the instructions and the continuous changes of its state takes Place as the program counter (PC) changes.
- Process is that executing unit of computation, which is controlled by some process (of the OS) for a scheduling mechanism that lets it execute on the CPU and by some process at OS for a resource management mechanism that lets it use the system-memory and other system resources such as network, file, display or printer.

Application program can be said to consist of number of processes

Example - Mobile Phone Device embedded software

- Software highly complex.
- Number of functions, ISRs, processes threads, multiple physical and virtual device drivers, and several program objects that must be concurrently processed on a single processor.
- Voice encoding and convoluting process─ the device captures the spoken words through a speaker and generates the digital signals after analog to digital conversion, the digits are encoded and convoluted using a CODEC,

UNIT IV REAL TIME OPERATING SYSTEMS – PART - 1 9

Process Concepts

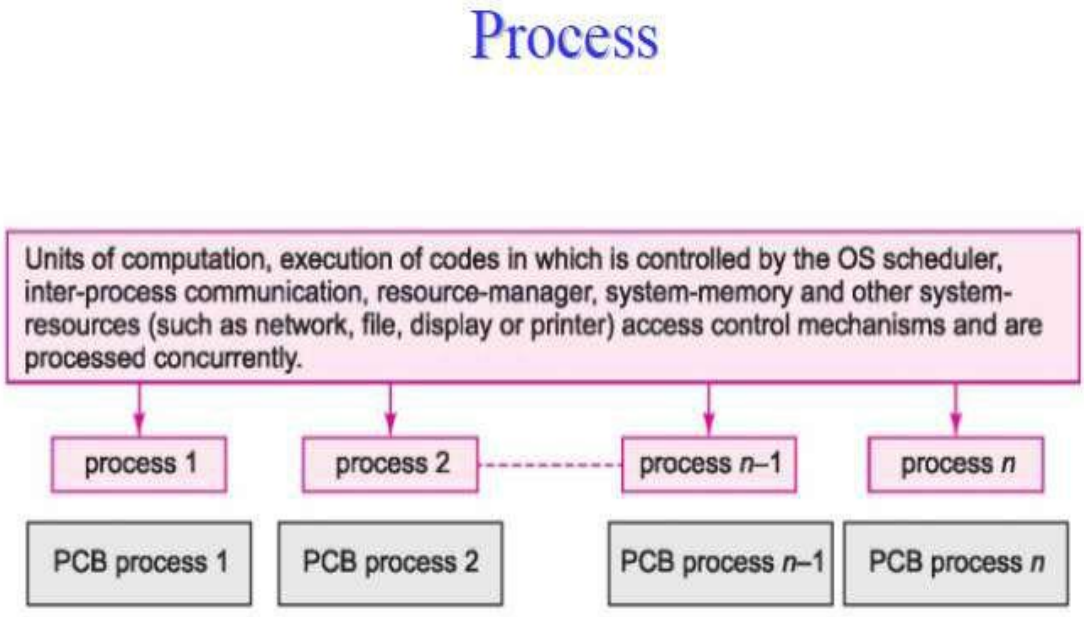
- A process consists of executable program (codes), *state* of which is controlled by OS, the *state* during running of a process represented by process-status (running, blocked, or finished), process structure—its data, objects and resources, and process control block (PCB).
- Runs when it is scheduled to run by the OS (kernel)
- OS gives the control of the CPU on a process's request (system call).
- Runs by executing the instructions and the continuous changes of its state takes Place as the program counter (PC) changes.
- Process is that executing unit of computation, which is controlled by some process (of the OS) for a scheduling mechanism that lets it execute on the CPU and by some process at OS for a resource management mechanism that lets it use the system-memory and other system resources such as network, file, display or printer.

Application program can be said to consist of number of processes

Example - Mobile Phone Device embedded software

- Software highly complex.
- Number of functions, ISRs, processes threads, multiple physical and virtual device drivers, and several program objects that must be concurrently processed on a single processor.
- Voice encoding and convoluting process─ the device captures the spoken words through a speaker and generates the digital signals after analog to digital conversion, the digits are encoded and convoluted using a CODEC,

- Modulating process,
- Display process,
- GUIs (graphic user interfaces), and
- Key input process ─ for provisioning of the user interrupts



Process Control Block

- A data structure having the information using which the OS controls the Process state.
- Stores in protected memory area of the kernel.
- Consists of the information about the process state

Information about the process state at Process Control Block…

- Process ID,
- process priority,
- parent process (if any),
- child process (if any), and
- address to the next process PCB which will run,
- allocated program memory address blocks in physical memory and in secondary (virtual) memory for the process-codes,
- allocated process-specific data addressblocks
- allocated process-heap (data generated during the program run) addresses,
- allocated process-stack addresses for the functions called during running of the process,
- allocated addresses of CPU register-save area as a process context represents by CPU registers, which include the program counter and stack pointer
- allocated addresses of CPU register-save area as a process context [Register-contents (define process context) include the program counter and stack pointer contents]
- process-state signal mask [when mask is set to 0 (active) the process is inhibited from running and when reset to 1, the process is allowed to run],

- Signals (messages) dispatch table [process IPC functions],
- OS allocated resources‘ descriptors (for example, file descriptors for open files, device descriptors for open (accessible) devices, device-buffer addresses and status, socket-descriptor for open socket), and
- Security restrictions and permissions.

Context

- Context loads into the CPU registers from memory when process starts running, and the registers save at the addresses of register-save area on the context switch to another process
- The present CPU registers, which include program counter and stack pointer are called context
- When context saves on the PCB pointed process-stack and register-save area addresses, then the running process stops.
- Other process context now loads and that process runs─ This means that the context has switched.

Threads and Tasks

Thread Concepts

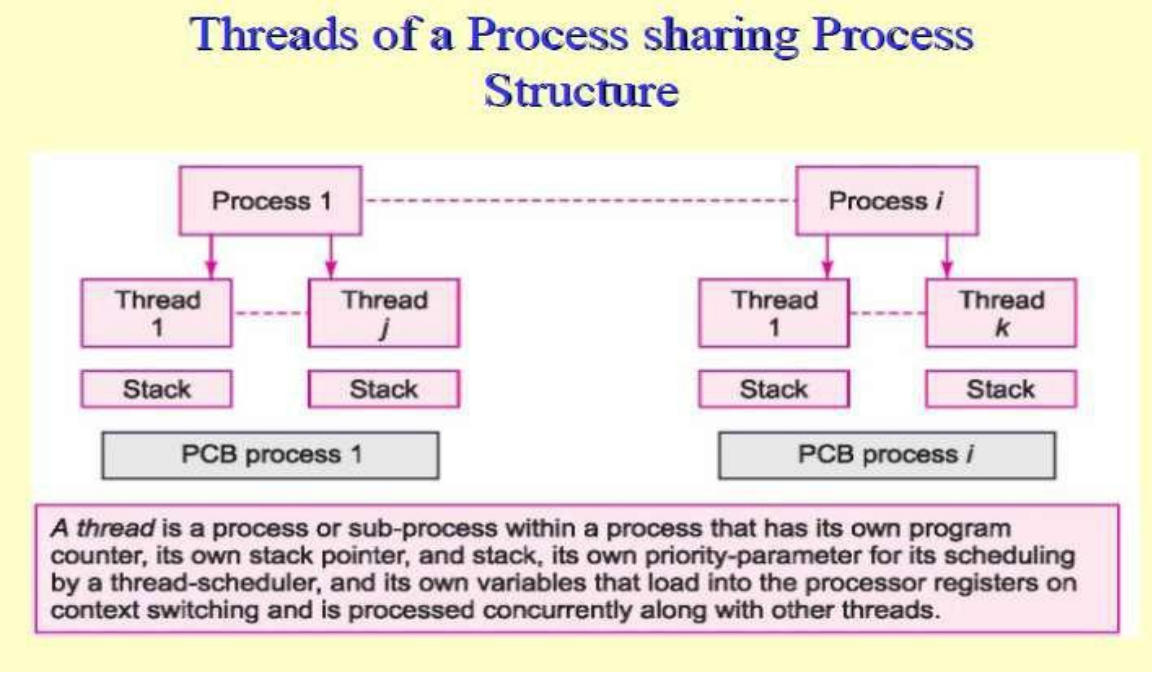
- A thread consists of executable program (codes), *state* of which is controlled by OS,
- The state information─ *thread-status* (running, blocked, or finished), *threadstructure*— its data, objects and a subset of the process resources, and *thread-stack.* Considered a lightweight process and a process level controlled entity.[Light weight means its running does not depend on system resources] .

Process… heavyweight

- Process considered as a heavyweight process and a kernel-level controlled entity.
- Process thus can have codes in secondary memory from which the pages can be swapped into the physical primary memory during running of the process. [Heavy weight means its running may depend on system resources]
- May have process structure with the virtual memory map, file descriptors, user–ID, etc.
- Can have multiple threads, which share the process structure thread
- *A* process or sub-process within a process that has its own program counter, its own stack pointer and stack, its own priority parameter for its scheduling by a thread scheduler
- Its‘ variables that load into the processor registers on context switching.
- Has own signal mask at the kernel. Thread‘s signal mask
- When unmasked lets the thread activate and run.
- When masked, the thread is put into a queue of pending threads.

Thread‘s Stack

- A thread stack is at a memory address block allocated by the OS.



Application program can be said to consist of number of threads or Processes:

Multiprocessing OS

- A multiprocessing OS runs more than one processes.
- When a process consists of multiple threads, it is called multithreaded process.
- A thread can be considered as daughter process.
- A thread defines a minimum unit of a multithreaded process that an OS schedules onto the CPU and allocates other system resources.

Thread parameters

- Each thread has independent parameters ID, priority, program counter, stack pointer, CPU registers and its present status.
- Thread states─ starting, running, blocked (sleep) and finished

Thread's stack

- When a function in a thread in OS is called, the calling function state is placed on the stack top.
- When there is return the calling function takes the state information from the stack top
- A data structure having the information using which the OS controls the thread state.
- Stores in protected memory area of the kernel.
- Consists of the information about the thread state

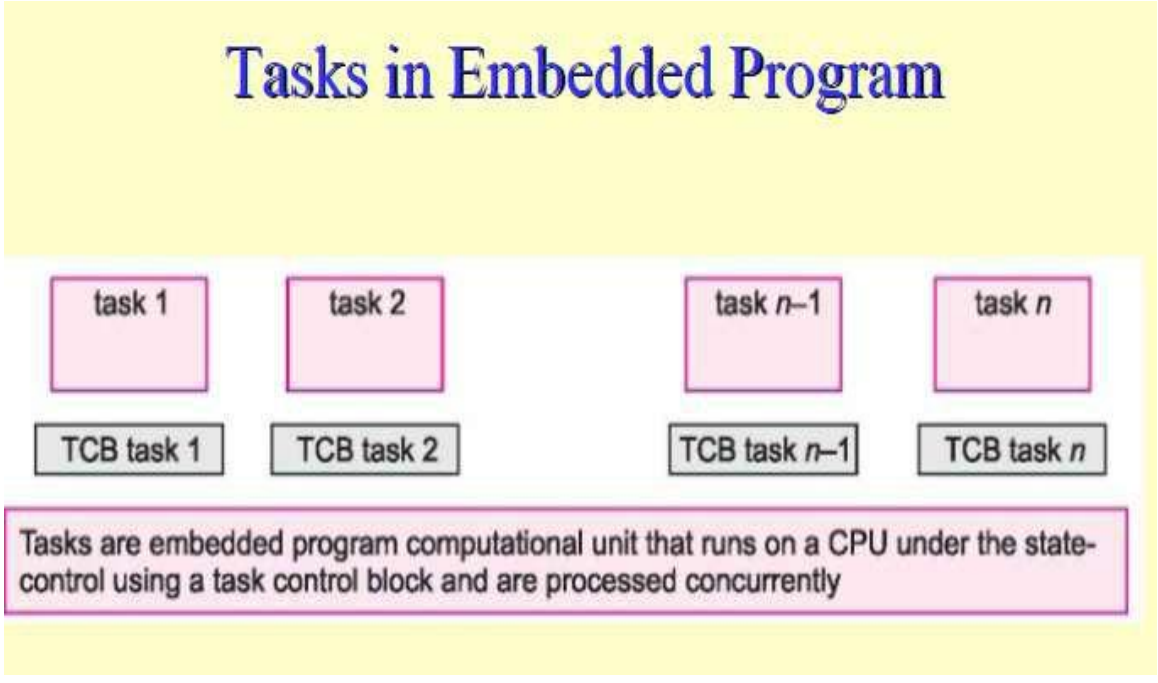
Thread and Task

- Thread is a concept used in Java or Unix.
- A thread can either be a sub-process within a process or a process within an application program.
- To schedule the multiple processes, there is the concept of forming thread groups and thread libraries.
- A task is a process and the OS does the multitasking.
- Task is a kernel-controlled entity while thread is a process-controlled entity.
- A thread does not call another thread to run. A task also does not directly call another task to run.
- Multithreading needs a thread-scheduler. Multitasking also needs a task-scheduler.
- *There may or may not be task groups* and task libraries in a given OS

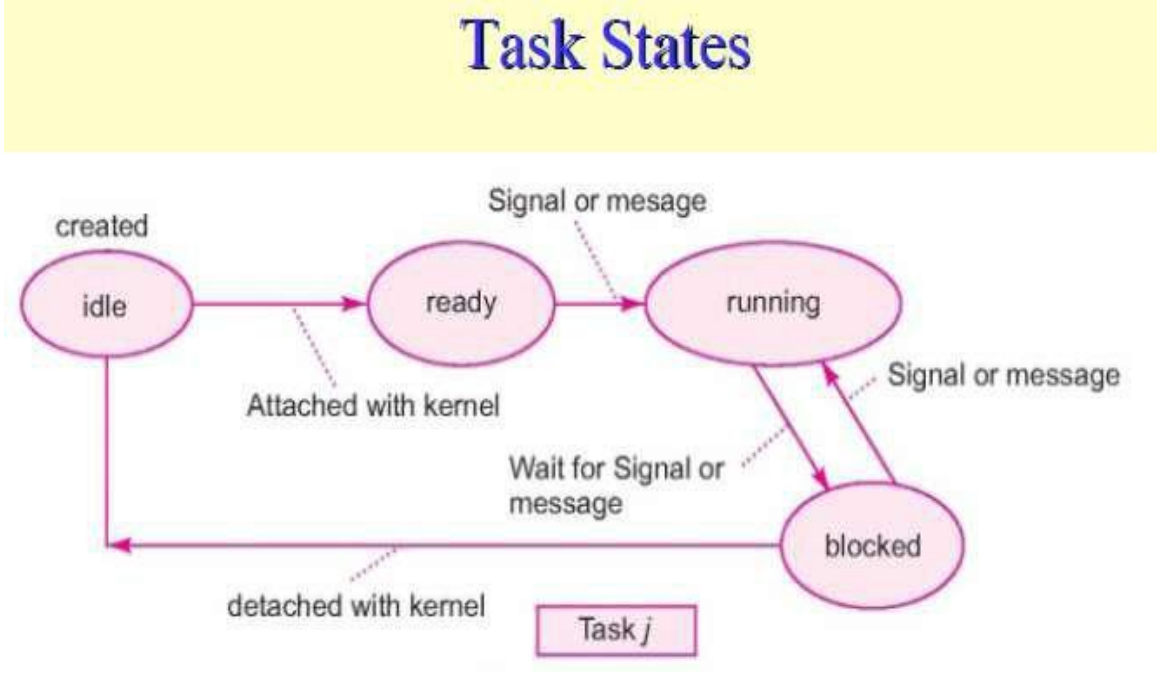
Task and Task States

Task Concepts

- An application program can also be said to be a program consisting of the tasks and task behaviors in various states that are controlled by OS.
- A task is like a process or thread in an OS.
- Task─ term used for the process in the RTOSes for the embedded systems. For example, VxWorks and μCOS-II are the RTOSes, which use the term task.
- A task consists of executable program (codes), *state* of which is controlled by OS, the *state* during running of a task represented by information of process status (running, blocked, or finished),process-structure—its data, objects and resources, and task control block (PCB).
- Runs when it is scheduled to run by the OS (kernel), which gives the control of the CPU on a task request (system call) or a message.
- Runs by executing the instructions and the continuous changes of its state takes place as the program counter (PC) changes.
- Task is that executing unit of computation, which is controlled by some process at the OS scheduling mechanism, which lets it execute on the CPU and by some process at OS for a resource-management mechanism that lets it use the system memory and other system-resources such as network, file, display or printer.
- A task─ an independent process.
- No task can call another task. [It is unlike a C (or C++) function, which can call another function.]
- The task─ can send signal (s) or message(s) that can let another task run.
- The OS can only block a running task and let another task gain access of CPU to run the servicing codes



Task States
(i) Idle state [Not attached or
not registered]
(ii) Ready State [Attached or registered]
(iii) Running state
(iv) Blocked (waiting) state
(v) Delayed for a preset period



Idle (created) state

- The task has been created and memory allotted to its structure however, it is not ready and is not schedulable by kernel.

Ready (Active) State

- The created task is ready and is schedulable by the kernel but not running at present as another higher priority task is scheduled to run and gets the system resources at this instance.

Running state

- Executing the codes and getting the system resources at this instance. It will run till it needs some IPC (input) or wait for an event or till it gets pre-empted by another higher priority task than this one.
-

Blocked (waiting) state

- Execution of task codes suspends after saving the needed parameters into its Context. It needs some IPC (input) or it needs to wait for an event or wait for higher priority task to block to enable running after blocking.

Deleted (finished) state

- Deleted Task─ The created task has memory deallotted to its structure. It frees the memory. Task has to be re-created.

Function

- Function is an entity used in any program, function, task or thread for performing specific set of actions when called and on finishing the action the control returns to the function calling entity (a calling function or task or process or thread).
- Each function has an ID (name)
- has program counter and
- has its stack, which saves when it calls another function and the stack restores on return to the caller.
- Functions can be nested. One function call another, that can call another, and so on and later the return is in reverse order

Memory Management Functions

Memory allocation

- when a process is created, the memory manager allocates the memory addresses (blocks) to it by mapping the process address space.
- Threads of a process share the memory space of the process

- Memory manager of the OS─ secure, robust and well protected.
- No memory leaks and stack overflows
- Memory leaks means attempts to write in the memory block not allocated to a process or data structure.
- Stack overflow means that the stack exceeding the allocated memory block(s)

Memory Management after Initial Allocation

Memory Managing Strategy for a system

- Fixed-blocks allocation
- Dynamic -blocks Allocation

- Dynamic Page-Allocation
- Dynamic Data memory Allocation
- Dynamic address-relocation
- Multiprocessor Memory Allocation
- Memory Protection to OS functions

Memory allocation in RTOSes

- RTOS may disable the support to the dynamic block allocation, MMU support to dynamic page allocation and dynamic binding as this increases the latency of servicing the tasks and ISRs.
- RTOS may not support to memory protection of the OS functions, as this increases the latency of servicing the tasks and ISRs.
- User functions are then can run in kernel space and run like kernel functions
- RTOS may provide for disabling of the support to memory protection among the tasks as this increases the memory requirement for each task

Memory Manager functions

(i) use of memory address space by a process,
(ii) specific mechanisms to share the memory space and
(iii) specific mechanisms to restrict sharing of a given memory space
(iv) optimization of the access periods of a memory by using an hierarchy of memory (caches, primary and external secondary magnetic and optical memories).

Remember that the access periods are in the following increasing order: caches, primary and external secondary magnetic and then or optical.

Fragmentation Memory
Allocation Problems
Fragmented not continuous memory addresses in two blocks of a process

- Time is spent in first locating next free memory address before allocating that to the process.
- A standard memory allocation scheme is to scan a linked list of indeterminate length to find a suitable free memory block.
- When one allotted block of memory is deallocated, the time is spent in first locating next allocated memory block before deallocating that to the process.
- the time for allocation and de-allocation of the memory and blocks are variable (not deterministic) when the block sizes are variable and when the memory is fragmented.
- In RTOS, this leads to unpredicatble task performance

Memory management Example

RTOS COS-II

- Memory partitioning
- A task must create a memory partition or several memory partitions by using function OSMemCreate ( )
- Then the task is permitted to use the partition or partitions.
- A partition has several memory blocks.
- Task consists of several fixed size memory blocks.
- The fixed size memory blocks allocation and de-allocation time takes fixed time (deterministic).
- OSMemGet ( )

– to provide a task a memory block or blocks from the partition

- OSMemPut ( )

─ to release a memory block or blocks to the partition

Interrupt Service routine

- ISR is a function called on an interrupt from an interrupting source.
- Further unlike a function, the ISR can have hardware and software assigned priorities.
- Further unlike a function, the ISR can have mask, which inhibits execution on the event, when mask is set and enables execution when mask reset.

Task

- Task defined as an executing computational unit that processes on a CPU and state of which is under the control of kernel of an operating system.

Distinction Between Function, ISR and Task

Uses

- Function─ for running specific set of codes for performing a specific set of actions as per the arguments passed to it
- ISR─ for running on an event specific set of codes for performing a specific set of actions for servicing the interrupt call.
- Task ─ for running codes on context switching to it by OS and the codes can be in endless loop for the event (s)

Calling Source

- Function─ call from another function or process or thread or task.
- ISR─ interrupt-call for running an ISR can be from hardware or software at any Instance.
- Task ─ A call to run the task is from the system (RTOS). RTOS can let another higher priority task execute after blocking the present one. It is the RTOS (kernel) only that controls the task scheduling.

Context Saving

- Function─ run by change in program counter instantaneous value. There is a stack. On the top of which the program counter value (for the code left without running) and other values (called functions‘ context) save.
- All function have a common stack in order to support the nesting
- ISR─ Each ISR is an event-driven function code. The code run by change in program counters instantaneous value. ISR has a stack for the program counter instantaneous value and other values that must save.
- All ISRs can have common stack in case the OS supports nesting
- Task ─ Each task has a distinct task stack at distinct memory block for the context (program counter instantaneous value and other CPU register values in task control block) that must save .
- Each task has a distinct process structure (TCB) for it at distinct memory block

Response and Synchronization

- Function─ nesting of one another, a hardware mechanism for sequential nested mode synchronization between the functions directly without control of scheduler or OS
- ISR─ a hardware mechanism for responding to an interrupt for the interrupt source calls, according to the given OS kernel feature a synchronizing mechanism for the ISRs, and that can be nesting support by the OS.
- ISR─ a hardware mechanism for responding to an interrupt for the interrupt source calls, according to the given OS kernel feature a synchronizing mechanism for the ISRs,and that can be nesting support by the OS

Structure

- Function─ can be the subunit of a process or thread or task or ISR or subunit of another function.
- ISR─ Can be considered as a function, which runs on an event at the interrupting source.
- A pending interrupt is scheduled to run using an interrupt handling mechanism in the OS, the mechanism can be priority based scheduling.
- The system, during running of an ISR, can let another higher priority ISR run.
- Task ─ is independent and can be considered as a function, which is called to run by the OS scheduler using a context switching and task scheduling mechanism of the OS.
- The system, during running of a task, can let another higher priority task run. The kernel manages the tasks scheduling

Global Variables Use

- Function─ can change the global variables. The interrupts must be disabled and after finishing use of global variable the interrupts are enabled.
- ISR─ When using a global variable in it, the interrupts must be disabled and after finishing use of global variable the interrupts are enabled (analogous to case of a function).
- Task ─ When using a global variable, either the interrupts are disabled and after finishing use of global variable the interrupts are enabled or use of the semaphores or lock functions in critical sections, which can use global variables and memory buffers.

Posting and Sending Parameters

- Function─ can get the parameters and messages through the arguments passed to it or global variables the references to which are made by it. Function returns the results of the Operations.
- ISR─ using IPC functions can send (post) the signals, tokens or messages. ISR can‘t use the mutex protection of the critical sections by wait for the signals, tokens or messages.

- Task ─ can send (post) the signals and messages.
- can wait for the signals and messages using the IPC functions, can use the mutex or lock protection of the code section by wait for the token or lock at the section beginning and messages and post the token or unlock at the section end.

Semaphore as an event signalling variable or notifying variable

- Suppose that there are two trains.
- Assume that they use an identical track.
- When the first train A is to start on the track, a signal or token for A is set (true, taken) and
- same signal or token for other train, B is reset (false, not released).

OS Functions for Semaphore as an event signalling variable or notifying variable:

- OS Functions provide for the use of a semaphore for signalling or notifying of certain action or notifying the acceptance of the notice or signal.
- Let a binary Boolean variable, *s*, represents the semaphore. The taken and post operations on *s*— *(i)*signals or notifies operations for communicating the occurrence of an event and (ii) for communicating taking note of the event.
- Notifying variable *s* is like a token — (i) acceptance of the token is taking note of that event (ii) Release of a token is the occurrence of an event

Binary Semaphore

- Let the token (flag for event occurrence) *s* initial value = 0
- Assume that the *s* increments from 0 to 1 for signalling or notifying occurrence of an event from a section of codes in a task or thread.
- When the event is taken note by section in another task waiting for that event, the *s* decrements from 1 to 0 and the waiting task codes start another action.
- When s = 1— assumed that it has been released (or sent or posted) and no task code section has taken it yet.
- When s = 0 — assumed that it has been taken (or accepted) and other task code
- section has not taken it yet

Binary Semaphore use in ISR and Task

- An ISR can release a token.
- A task can release the token as well accept the token or wait for taking the token

Device Management Functions

Number of device driver ISRs in a system,

Each device or device function having s a separate driver, which is as per its hardware

Software that manages the device drivers of each device

Provides and executes the modules for managing the devices and their drivers ISRs.

effectively operates and adopts appropriate strategy for obtaining optimal performance for the devices.

Coordinates between application-process, driver and device-controller.

Device manager

Process sends a request to the driver by an interrupt; and the driver provides the actions by executing an ISR.

Device manager polls the requests at the devices and the actions occur as per their priorities.

Manages IO Interrupts (requests) queues.

creates an appropriate kernel interface and API and that activates the control register specific actions of the device. [Activates device controller through the API and kernel interface.]

manages the physical as well as virtual devices like the pipes and sockets through a common strategy.

Device management has three standard approaches

Three types of device drivers:

(i) Programmed I/Os by polling from each device its the service need from each device.

(ii) Interrupt(s) from the device drivers' device- ISR and

(iii) Device uses DMA operation used by the devices to access the memory.

Most common is the use of device driver ISRs

Device Manager Functions

Device Detection and Addition

Device Deletion

Device Allocation and

Registration

Detaching and Deregistration

Restricting Device to a specific process

Device Sharing

Device control

Device Access Management

Device Buffer Management

Device Queue, Circular-queue or blocks of queues Management

Device drivers updating and upload of new device-functions

Backup and restoration

Device Types

char devices and

block devices

Set of Command Functions for the Device
Management Commands for Device

create

open

write

read

ioctl

close and

delete

IO control Command for Device

(*i*) Accessing specific partition information

(*ii*) Defining commands and control functions of device registers

(*iii*) IO channel control

Three arguments in ioctl ( )

First Argument: Defines the chosen device and its function by passing as argument the device descriptor (a number), for example, fd or sfd Example is fd = 1 for read, fd = 2 for write.

Second Argument: Defines the control option or uses option for the IO device, for example, baud rate or other parameter optional function

Third Argument: Values needed by the defined function are at the third argument

Example

Status = ioctl (fd, FIOBAUDRATE, 19200) is an instruction in RTOS VxWorks.

fd is the device descriptor (an integer returned when the device is opened)

FIOBAUDRATE is the function that takes value = 19200 from the argument. This at configures the device for operation at 19200-baud rate.

Device Driver ISR functions

ISR functions

intlock ( ) to disable device-interrupts systems,

intUnlock ( ) to enable device-interrupts,

intConnect ( ) to connect a C function to an interrupt vector

Interrupt vector address for a device ISR points to its specified C function.

intContext ( ) finds whether interrupt is called when an ISR was in execution

Unix OS functions

UNIX Device driver functions

Facilitates that for devices and files have an analogous implementation as far as possible.

open ( ),

close ( ),

read ( ),

write ( ) functions analogous to a file *open*,

close, *read* and *write* functions.

APIs and kernel interfaces in BSD (Berkley sockets for devices)

open,

close,

read

write

in-kernel commands

(i) select ( ) to check whther read/write will succeed and then select

(*ii*) *ioctl* ( )

(*iii*) *stop* ( ) to cancel the output activity from the device.
(iv) *strategy* ( ) to permit a block *read or write* or character *read or write*

Round Robin Time Slicing of tasks of equal priorities
Common scheduling models

- Cooperative Scheduling of ready tasks in a circular queue. It closely relates to function queue scheduling.
- Cooperative Scheduling with Precedence Constraints
- Cyclic scheduling of periodic tasks and Round Robin Time Slicing Scheduling of equal priority tasks
- Preemptive Scheduling
- Scheduling using 'Earliest Deadline First' (EDF) precedence.

Common scheduling models

- Rate Monotonic Scheduling using _higher rate of events occurrence First‘ precedence
- Fixed Times Scheduling
- Scheduling of Periodic, sporadic and aperiodic Tasks
- Advanced scheduling algorithms using the probabilistic Timed Petri nets (Stochastic) or Multi Thread Graph for the multiprocessors and complex distributed systems.

Round Robin Time Slice Scheduling of Equal Priority Tasks
Equal Priority Tasks

- Round robin means that each ready task runs turn by in turn only in a cyclic queue for a limited time slice.
- Widely used model in traditional OS.
- Round robin is a hybrid model of clock-driven
- model (for example cyclic model) as well as event driven (for example, preemptive)
- A real time system responds to the event within a bound time limit and within an explicit time.

Tasks programs contexts at the five instances in the Time Scheduling Scheduler for C1 to C5
Programming model for the Cooperative Time sliced scheduling of the tasks

Program counter assignments on the scheduler call to tasks at two consecutive time slices. Each cycle takes time = N tslice

Case : Tcycle = N Tslice

- Same for every task = Tcycle
- Tcycle ={Tslice)} N + tISR.
- tISR is the sum of all execution times for the ISRs
- For an i-th task, switching time from one task to another be is *st* and task execution time be is *et*
- Number of tasks = N

Worst-case latency

- Same for every task in the ready list
- Tworst = {N (Tslice)} + tISR.
- tISR is the sum of all execution times for the ISRs
- i = 1, 2, …, N 1 , N

VoIP Tasks Example

- Assume a VoIP [Voice Over IP.] router.