DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EE6602- EMBEDDED SYSTEMS

QUESTION BANK

UNIT I - INTRODUCTION TO EMBEDDED SYSTEMS

PART A

1. Define system.

A system is a way of working, organizing or doing one or series of tasks by following a fixed plan, program and set of rules.

2.What are the typical characteristics of an embedded system?)(apr/may2017)

T ypical characteristics:perform a single or tightly knit set of functions; increasingly high-performance & real-time constrained; power, cost and reliability are often important attributes that influence design;Application specific processor design can be a significant component of some embedded systems.

3. What is the need of watchdog timer?(may/june 2016)

Watchdog timer is a timer that resets the processor in case the program gets stuck for an unexpected length of time.

4.What are the challenges faced in designing an embedded system? (nov/dec 2016) (may/june 2016)

- Amount and type of hardware needed.
- Clock rate reduction
- Voltage reduction
- Process deadlines'
- Flexibility and upgrade ability
- Wait, stop and cache disable instructions.

5.When do we need an RTOS?

RTOS has a basic function of the OS plus functions for real time task scheduling and interrupt latency control. It uses the timers and system clocks, time allocation and de-allocation to attain best utilization of the CPU time under the given timing constraint for the task.

6.Name some processor in complex embedded system?

General purpose microprocessor
Microcontroller
    Single purpose
    Dual core processor

7.What are the abstractions steps in design process? (nov/dec 2016)(apr/may2017)

- Requirements
- Specifications
- Architecture
- Components
- System integration

8. What are the classification of ES?

- Small scale embedded system
- Medium scale embedded system
- Sophisticated embedded system

9.What are the tools used for programming tools are used in a complex software design?

- RTOS
- Source code engineering tool
- Simulator
- Debugger
- Assembler
- Integrated development environment

10.Name some hardware components used in embedded system.

- Power source
- Clock oscillator circuit
- Crystal resonator
- External oscillator IC
- Real time clock
- Memories

UNIT II - EMBEDDED NETWORKING
PART A

1. What is CAN bus?(apr/may2017)

It is a standard bus used at the control area network generally in automotive and industrial electronics.

2. What is COM port?

It is a port at the computer where a mouse, modem, serial printer or mobile serial printer connects for serial I/O in UART mode and there are handshaking signals for exchange of signals before UART mode communications.

3.Define device driver. (nov/dec 2016)(apr/may2017)

A device driver is a software for controlling receiving and sending byte or stream of bytes from or to a device.Device is a unit that has a processing element and that connects to the processor of embedded system internally or through the port or bus. It has fixed pre assigned port addresses according to its interfacing or bus controller circuit.

4. what is device decoder?

It is a circuit to take the system address bus signal as the input and generate a device select signals CS, for the port address selection during the device read or write instruction of the system processor.

5. What is an I/O port?

It is port for input or output operations at an instance. Handshake input and handshake output also known as I/O ports. For examples, a keypad is said to connect to an I/O port.

6. Define protocol.

Protocol is a way of transmitting messages on a network by using software for adding the additional bit such as starting bits, header, addresses of source and destination, error control bit and ending bits.

7. Define bus.

Bus is a set of parallel lines which carry signals from one to another unit. Bus enables interconnecting among many units in a simple way. The signal specific sequences according to a method or protocol.

8. What is the need of timing diagram?

Timing diagram reflects the relative time intervals of the signals on the external buses with respect to the processor clock pulses.

9.What are the types of architecture model?

- Single level architecture
- Two-level architecture
- Multi-level architecture

10.Mention some metrics of a bus.

- Simplifies number of interconnections compared to direct connections between one another.
- Provides a common way of interconnecting different or same type of I/O devices.
- Can add a new device or system interface that is compatible with a system I/O bus.

UNIT III - EMBEDDED FIRMWARE DEVELOPMENT ENVIRONMENT

PART A

1. What is EDLC?

Embedded produce development life cycle is an analysis design implementation based standard problem solving approach for embedded product development.

2. What are the objective of EDLC?

- Ensure that high quality products are delivered to end user.
- Risk minimization
- Maximize the productivity

3.What are the different phases of EDLC? Apr/may2017

1. Need
2. Conceptualization
3. Analysis
4. Design
5. Development and testing
6. Deployment
7. Support
8. Upgrades
9. Retirement

4.What is meant by conceptualization?

It is the product concept development phase and it defines the scope of the concept, performs cost benefits analysis and feasibility study and prepare project management and risk management.

5. What is upgrade phase?

The upgrade phase of product development deals with the development of upgrades for the product which is already present in the market.

6. What is prototyping model?

Prototyping model is a EDLC model which is the variation of the iterative model in which a more refined prototype is produced at the end of each iteration.

7. What is DFG model?

The data flow graph model translates the data processing requirements into a flow graph. It is data driven model in which the program execution is determined by data.

8. What is sequential program model?

In sequential programming model, the functions or processing requirements are executed in sequence. It is same as the conventional procedural programming.

9. What is object oriented model?

It is an object based model for modeling system requirements. It divides complex software requirements into simple well defined pieces called objects.

10. What is state machine model?nov/dec2016

The state machine model contain a number of states are finite. In other words the system is described using a finite number of possible states.

UNIT IV - RTOS BASED EMBEDDED SYSTEM DESIGN
PART A

1. What is an operating system?

The operating system acts as a bridge between the user applications and the underlying system resources through a set of system functionalities and services.

2.What is preemptive and nonpreemptive scheduling?nov/dec2016

Under nonpreemptive scheduling once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or switching to the waiting state. Preemptive scheduling can preempt a process which is utilizing the CPU in between its execution and give the CPU to another process.

3. What is thread in the operating system context?may/june2016

A thread is the primitive that can execute code. A thread is a single sequential flow of control within a process. Thread is also known as light weight process.

4.What are the various multi tasking models? May/june2016

- Co-operative multitasking
- Preemptive multitasking
- Non-preemptive multitasking

5.What is co-operative multitasking?

It is a multitasking model in which a task/process gets a chance when the currently executing task relinquishes the CPU voluntarily.

6.What are the different queues associated with process scheduling?

- Job queues
- Ready queues
- Device queues

7.What are the types of non-preemptive scheduling?

1. First come first served
2. Last come last served
3. Shortest job first
4. Priority based

8. What is pipe?

Pipe is a selection of the shared memory used by processes for communicating. Pipes follow the client server architecture.

9. What is IPC?

The mechanism through which processes/task communicate each other is known as inter process communications.
It is essential for process co-ordination.

10. What is deadlock?

Deadlock is a situation where non of the processes are able to make any progress in their execution. It is the condition in which a process is waiting for a resources held by another process which is waiting for a resource held by the first process.Livelock is a condition where a process always does something but is unable tp make any progress in the execution completion

UNIT V - EMBEDDED SYSTEM APPLICATION DEVELOPMENT

PART A

1. Define MUCOS.

MUCOS is a portable, ROMable, Scalable , real time and multitasking kernel. It is used over thousands of applications including automotive , avionics, consumer electronics, medical devices, military aerospace networking.

- Cameras
- Automotive
- Medical devices
- Aerospace

2.What is AVCM?

Automatic chocolate vending machine is a machine using which the children can automatically purchase the chocolates. The payment is made by inserting the coins of appropriate amount into a win slot.

3.What are the three phases of washing machine?

- Wash phase
- Spin phase
- Rinse phase

4.What is meant by ECU?

Automotive embedded system are normally built around microcontroller or DSP or a hybrid of the two and are generally known as electronic control units.

5.What are the classifications of ESU?

- High speed electronic control unit
- Low speed electronic control unit

6.What is the use of HECU?

High speed electronic control unit are deployed in critical control units requiring fast response like fuel injection system, antilock brake system.

7. What is the use of LEC?

Low speed electronic control unit are deployed in applications where response time is not so critical. They are generally built around low cost microprocessor and DSP. Audio controller , passenger and driver door lock are the examples of LEC.

8.Name some serial buses used in automotive communication.apr/may2017

- Controller area network
- Local interconnect network
- Media oriented system transport bus

9.What are the key players of the automotive embedded marker?

- Silicon providers
- Tools and platform providers
- Solution providers

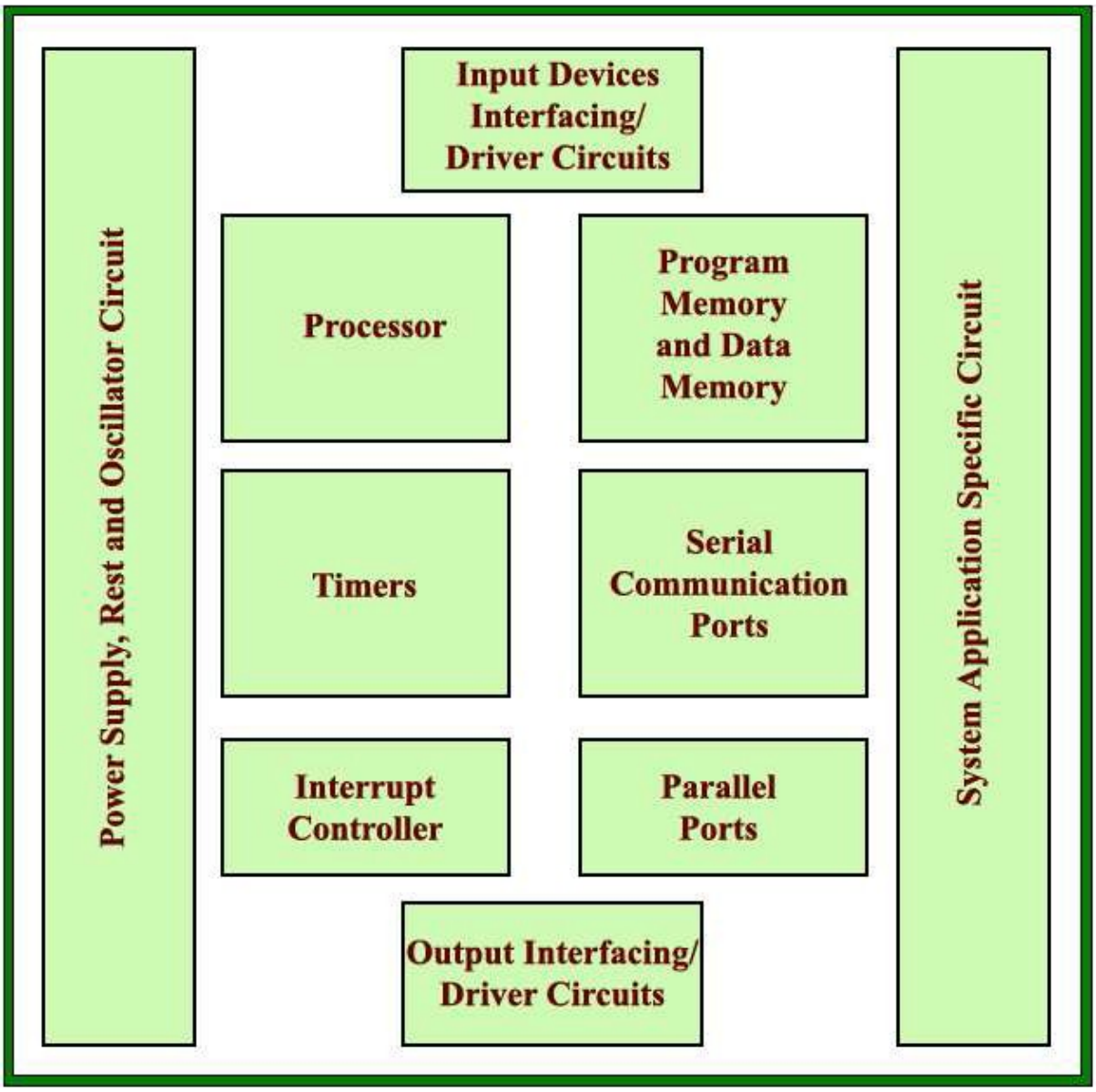
10.What are the applications of an embedded system?nov/dec2016, apr/may2017

1. Consumer electronics, e.g., cameras, camcorders,
2. Consumer products, e.g., washers, microwave ovens,
3. Automobiles (anti-lock braking, engine control,
4. Industrial process controllers & avionics/defense

UNIT-1

1. Discuss about the structural units in embedded processor and how a processor is selected for an embedded application?

EMBEDDED SYSTEM CONSTRAINTS:



Processor
Processor is the heart of the embedded system. It consists of two units:
Two Essential Units: Operations
Control Unit (CU), Fetch
Execution Unit (EU) Execute- It includes ALU and executes the program task, say, halt, interrupt
and jump or another set of instructions

Processor runs the cycle of fetch and execute. Processor mostly in the form of IC or in the form of
Core ASIP [Application Specific Instruction Processor] or Soc [System on Chip], core means a part of
functional circuit on VLSI chip.

2. With a neat diagram, explain the working of Direct Memory Access (DMA)?

Embedded hardware components

- Microcontroller or ASIP (Application Specific Instruction Set Processor)
- RAM for temporary variables and stack
- ROM for application codes and RTOS codes for scheduling the tasks
- EEPROM for storing user data, user address, user identification codes, card number and expiry date
- Timer and Interrupt controller
- A carrier frequency ~16 MHz generating circuit and Amplitude Shifted Key (ASK)
- Interfacing circuit for the I/Os
- Charge

Embedded Software

- Boot-up, Initialisation and OS programs
- Smart card secure file system
- Connection establishment and termination
- Communication with host
- Cryptography
- Host authentication
- Card authentication
- Addition parameters or recent new data sent by the host (for example, present balance left)

Smart Card OS Special features

- Protected environment.
- Every method, class and run time library should be scalable.
- Code-size generated be optimum.
- Memory should not exceed 64 kB memory.
- Limiting uses of specific data types; multidimensional arrays, long 64-bit integer and floating

pointsSmart Card OS Limiting features _ Limiting uses of the error handlers, exceptions, signals,
serialization, debugging and profiling. [Serialization means process of converting an object is converted
into a data stream for transferring it to network or from one process to another. At receiver end there is
de-serialization.]

Smart Card OS File System and Classes

Three-layered

ROM image, Programming Languages and Program models
ROM Image
• Final stage software also called ROM image
* (Just as an image is a unique sequence and arrangement of pixels, embedded software is also a unique
placement and arrangement at each ROM address of bytes for instructions and data.)
System
ROM memory embedding the software, RTOS, data, and vector addresses
Final machine software
Bytes at each address defined for creating the ROM image.
By changing this image, the same hardware platform work differently and can be used for entirely

3.Explain the memory management method in detailed manner?
a. Functions Assigned to the ROM or EPROM or Flash
1. Storing 'Application' program from where the processor fetches the instruction codes
2. Storing codes for system booting, initializing, Initial input data and Strings.
3. Storing Codes for RTOS.
4. Storing Pointers (addresses) of various service routines.
b. Functions Assigned to the Internal, External and Buffer RAM
1. Storing the variables during program run,
2. Storing the stacks,
3. Storing input or output buffers for example, for speech or image .
c. Functions Assigned to the EEPROM or Flash
Storing non-volatile results of processing
d. Functions Assigned to the Caches
1. Storing copies of the instructions, data and branch-transfer instructions in advance from external memories and
2. Storing temporarily the results in write back caches during fast processing
(v) Interrupts Handler
Interrupt Handling element for the external port interrupts, IO interrupts, timer and RTC interrupts, software interrupts and Exceptions
(vi) Linking Embedded System Hardware
• Linking and interfacing circuit for the Buses by using the appropriate multiplexers, and decoders,
demultiplexers Interface the various
system units

4. Explain the I/O Communication ports in detailed manner?

a. Communication Driver(s) Network Ethernet or serial driver to communicate with host embedded system Expansion
Facility Serial Bus(es):
For example, UART (512 kbaud/s), 1-wire CAN (33 kbps),
Industrial I2C (100kbps), SM I2C Bus (100 kbps), SPI (100 kbps), Fault tolerant CAN (110 kbps),

Serial Port (230 kbps), MicroWire (300 kbps).
- SCSI parallel (40 Mbps), Fast SCSI (8M to 80 Mbps) , Ultra SCSI-3 (8M to 160 Mbps), FireWire/IEEE 1394 (400 Mbps, 72 meter), High Speed USB 2.0 (480 Mbps, 25 meter)
- Parallel Bus(es): PCI, PCI-X

b. Media IO Control Element
c. Keypad or Keyboard IO Interface
d. LCD Display System Interface
e. ADC – Single or Multi channel
f. DAC
g. GPIB Interface Element
h. Pulse Dialing Element
i. Modem and j. Bluetooth, 802.11, IrDA.

5.Explain the Basic Circuit Elements at the Embedded control System?

(i) Power Source

1. System own supply with separate supply rails for IOs, clock, basic processor and memory and
analog units, *or*
2. Supply from a system to which the embedded system interfaces, for example in a network card,
OR
Charge pump concept used in a system of little power needs, for examples, in the mouse or contactless
smart card

Power Dissipation Management

1. Clever real-time programming by Wait and Stop instructions
2. Clever reduction of the clock rate during specific set of instructions
3. Optimizing the codes and
4. Clever enabling and disabling of use of caches or cache blocks

(ii) Clock Oscillator Circuit and Clocking Units

1. Appropriate clock oscillator circuit
2. Real Time Clock*(System Clock) and Timers driving hardware and software

(iii) Reset Circuit

1. Reset on Power-up
2. External and Internal Reset circuit
3. Reset on Timeout of Watchdog timer

components of embedded system

- **It has Hardware**

Processor, Timers, Interrupt controller, I/O Devices, Memories, Ports, etc.

- **It has main Application Software**

Which may perform concurrently the series of tasks or multiple tasks.

- **It has Real Time Operating System (RTOS)**

RTOS defines the way the system work. Which supervise the application software. It sets the rules during the
execution of the application program. A small scale embedded system may not need an RTOS.

6.Discuss about the memory devices in clear manner?

Memory is an essential part of all microprocessor based systems. Different memory types are available to suit
different tasks. Some of these memory types will be discussed here.

random access memory (ram)

This is memory that can be written to and read from as often as we wish. The 'Random Access' refers
to the fact that we can access any position in the memory equally fast. RAM is used for short term storage, such
as for storing variables that are used by a program. RAM is volatile memory. It will only store contents (at best)
for as long as the power is applied.

ram signals

A typical RAM device will have (at least) the following set of signals:

Data Bus: This is a set of lines which carry the data into the memory or out of it. We refer to the width of the memory by how many bits can be written or read simultaneously.

Address Bus: This is a set of lines which is used to specify the location of the data to be read/written.

WR line: This line tells the RAM it is going to be written to or read from.

OE line: This line enables the data bus as output for when we read data from the device.

CS line: This line is used to select the device. This line enables or disables the entire chip.

Typical timing diagrams may be found in the lab sheet.

Basic RAM Types

i. dram

Dynamic RAM uses tiny capacitors to store each bit of information. This is a cheap, high density
technology. The capacitors need to be recharged periodically and so DRAM needs to be refreshed frequently.
DRAM forms the basis for most computer memory. Usually DRAM is used together with a DRAM controller
which handles the refresh cycles.

ii. sram

Static RAM uses a flip flop to store each bit in the memory. Because the data is stored in a proper flip
flop there is no need to refresh SRAM. SRAM is more expensive and lower density than DRAM.

7. Explain any one of the processor in detailed manner (ASSP)?

• ASSP is dedicated to specific tasks and provides a faster solution.
• An ASSP is used as an additional processing unit for running the application in place of using embedded
software.

Examples: IIM7100, W3100A

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

Single purpose processor or Application Specific Instruction processor

• 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

Use of Accelerator Cores:

Examples

Java Accelerator *Nazonin Communications* Java codes run 15 to 60 Times fast,
Video Accelerator for fast Video Processing

Multi core processors or multiprocessor system using GPPs

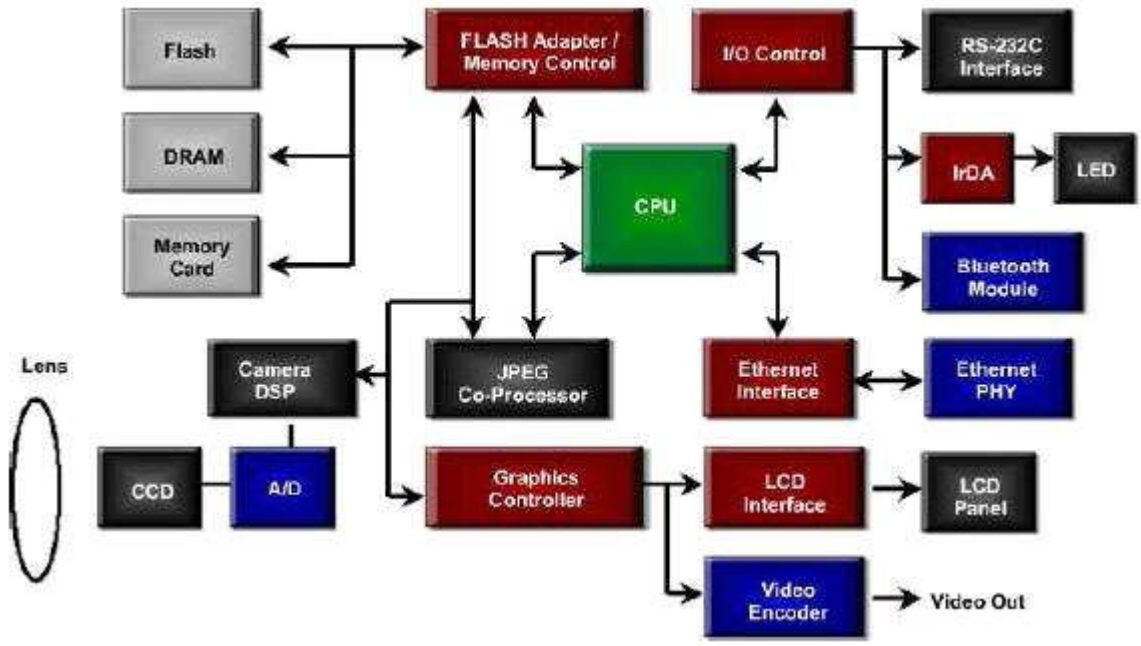
Examples

• Multiprocessor system for Real time performance in a video-conference system,
• Embedded firewall cum router,
• High-end cell phone.etc.

MOORE'S LAW

• Moore's law describes a long-term trend in the history of computing hardware.
• Since the invention of the integrated circuit in 1958, the number of transistors that can be placed
inexpensively on an integrated circuit has increased exponentially, doubling approximately every two
years.
• The trend was first observed by Intel co-founder Gordon E. Moore in 1965.
• Almost every measure of the capabilities of digital electronic devices is linked to Moore's law: processing
speed, memory capacity, etc.

8.BASIC COMPONENTS OF EMBEDDED SYSTEMS



Design Issues

- Cost
- I/O capability
- Size
- Power consumption
- On-Chip memory
- Performance
- Software
- Instruction set
- Development tools
- Testability and reliability
- Analog components

  Sensors, controllers,
- Digital components

  Processor, coprocessors

  Memories

  Controllers, busses

  ASIC: ***A****pplication-****S****pecific* ***I****ntegrated* ***C****ircuit, a chip designed*
- *for a particular application*
- Converters – A/D, D/A, …
- Software

  Application programs

  Exception handlers

Design Metrics

- Unit cost – the $ cost for each unit excluding development cost
- NRE cost: $ cost for design and development
- Size: The physical space reqd. – determined by bytes of sw, number of gates and transistors in hw
- Performance: execution time or throughput of the system
- Power: lifetime of battery, cooling provisions
- Flexibility: ability to change functionality without heavy NRE cost

Application Specific Characteristics

• Application is known before the system is designed
• System is however made programmable for
– Feature upgrades
– Product differentiation
• Often application development occurs in parallel to system development
– Hw-Sw partitioning should be as delayed as possible
• For upgrades design reuse is an important criterion

9.Detailed about the Programmable Logic Technologies?

Programmable logic is used to build up complex circuits without the need to connect up many gates.
They consist of chips which contain many logic gates. The way in which these gates are connected is
determined by the way in which the device is configured (programmed).
Programmable logic is commonly used because it is more compact and generally can run faster than a
circuit made out of discrete gates. Programmable logic offers design security.
There has been remarkable progress in the development of programmable logic devices over the last
few years. Because of the vast changes that the industry has seen there are very few standards in terms ofsignals and compatibility between devices.

PAL (Programmable Array Logic) and PLA (Programmable Array Logic)

These were the earliest form of programmable logic. They had a simple grid type structure which was
programmed by blowing fuses. They were only programmable once. The programming was done in a simple
language, typically PALASM, ABEL or CUPL.

GAL (Generic Array Logic)

Slightly more complex and flexible structure than PAL. Reprogrammable! They typically had a
JTAG (Joint Test and Access Group) interface for programming. This is an industry standard (IEEE1149.1)
four wire interface that has become one of the dominant programming and debugging interfaces for
programmable logic, microcontrollers and larger processors.

PEEL (programmable electrically erasable logic)

Very similar to GALs, but made of CMOS rather than bipolar technology.

CPLD (Complex Programmable Logic Device)

These devices are typically made up of many small PAL's connected together with a programmable
interconnection system. They are much more flexible than any of the previous programmable logic families.

FPGA (Field Programmable Gate Array)

These are the high end of the market. They consist of lots of programmable logic blocks. These blocks
are each quite complex, typically consisting of PAL type logic as well as devices such as configurable flip flops.
They are available in sizes up to several million gates.
They often have peripherals such as phase locked loop clock multipliers, memory blocks and communication (LVDS) systems.
There are many FPGA's which are big enough to hold entire microprocessors. These are becomingmore popular. They are called “soft cores”.

10.Detailed about the Watchdog timer in clear manner?

A timing device such that it is set for a preset time interval and an event must occur during that interval else the device will generate the timeout signal on failure to get that event in the watched time interval. On that event, the watchdog timer is disabled to disable generation of timeout or reset Timeout may result in processor start a service routine or start from beginning

Watchdog timer application

• An application in mobile phone is that display is off in case no GUI interaction takes place within a watched time interval.
• The interval is usually set at 15 s, 20 s, 25 s, 30 s in mobile phone.
• This saves power. An application in temperature controller is that if controller takes no action to switch off the current within preset watched time interval, the current is switched off and warning signal is raised as indication of controller failure. Failure to switch off current may burst a boiler in which water is heated.

Provisioning of watchdog timer

A software task can also be programmed as a watchdog timer Microcontroller may also provide for a watchdog timer.
68HC11 microcontroller watchdog timer 68HC11 microcontroller watchdog timer There are two registers, CONFIG (system configuration control register) and COPRST (computer operating properly and processor reset on failure).

They are for programming the interrupts of the watchdog timer.CONFIG has a bit, NOCOP. It configures when processor writes the configuration word at the address 0x003F. NOCOP is the 2nd bit of CONFIG If NOCOP is reset to 0 the COP facility is enabled. [COP means computer (68HC11 watchdog timer operating properly]. COP facility provides for keeping a watch on the user program execution timeWatchdog timer overflows (time outs) Watchdog timer overflows (time outs) 68HC11 program counter is reset according to the 16 bits (lower and higher bytes) preloaded at the addresses 0xFFFA and 0xFFFB, respectively.

If these 16 bits are the same as the bits in 0xFFFE and 0xFFFF, then the microcontroller executes instructions as when it resets on power up or else executes the routine at the 16-bit address fetched from 0xFFFE and 0xFFFF on failure within the watched time interval. A watchdog timer number of applications • It is timing device such that it is set

for a preset time interval and an event must occur during that interval and program instuction must disable the watchdog timer else the device will generate on the timeout signal an interrupt for the failure to get that event in the watched time interval.

UNIT II

1.Write short notes about the I2C in detailed manner?

The Bus has two lines that carry its signals— one line is for the clock and one is for bi-directional data. There is a standard protocol for the I 2C bus.

Device Addresses and Master in the I Device Addresses and Master in the I2C bus Each device has a 7-bit address using which the data transfers take place. Master can address 127 other slaves at an instance. Master has at a processing element functioning as bus controller or a microcontroller with I 2C (Inter Integrated Circuit) bus interface circuit.

Slaves and Masters in the I Slaves and Masters in the I2C bus Each slave can also optionally has I 2C (Inter Integrated Circuit) bus controller and processing element. **Number of masters can be connected on the bus**.

However, at an instance, master is one, which initiates a data transfer on SDA (serial data) line and which transmits the SCL (serial clock) pulses. From master, a data frame has fields beginning from start bit

Synchronous Serial Bus Fields and its length

First field of 1 bit─ Start bit similar to one in an UART

Second field of 7 bits ─ address field. It defines the slave address, which is being sent the data frame (of many bytes) by the master

Third field of 1 control bit─ defines whether a read or write cycle is in progress

Fourth field of 1 control bit─ defines whether is the present data is an acknowledgment (from slave)

Fifth field of 8 bits ─ I 2C device data byte

Sixth field of 1-bit─ bit NACK (negative acknowledgement) from the receiver. If active then acknowledgment after a transfer is not needed from the slave, else acknowledgement is expected from the slave

Seventh field of 1 bit ─ stop bit like in an UART.

Disadvantage of I 2C bus

• Time taken by algorithm in the hardware that analyzes the bits through I 2C in case the slave hardware does not provide for the hardware that supports it.

• Certain ICs support the protocol and certain do not.

• Open collector drivers at the master need a pull-up resistance of 2.2 K on each line.

2.Write short notes about the RS232C in detailed manner?

RS - 232C

The TTL logic signals are not what is transmitted between equipment such as modems computers and terminals. One standard that describes the signals for communicating is the RS-232C standard.

This standard causes as much trouble as any standard you can name because it deals with two types of equipment, data communications equipment (DCE) and data terminal equipment (DTE).

• Typically, a modem is DCE and a terminal or computer is DTE.
• The problem is that the cable to connect DCE to DTE is different from the cable to connect DTE to DTE.
• Also, nonstandard use of the control signals is a problem.
• Murphy's law says that if you need to connect to a serial port, you always have the wrong cable.
• RS-232C specifies 25 signal pins with a – male DTE connector and – a female DCE connector.
• Most of these signals are not used in most applications so a 9 pin subset is used. – Actually in some cases, only three wires are used.
• The mark and space signals are inverted by the standard to the following levels: A logic high or mark is between -3 V and -15 V under load A logic low or space is between +3 V and +15 V under load.
• Typically, +12 and -12 volts are used.
• The voltage swing is > the TTL 5 volts for noise immunity.
RS-232C interface TTL levels RS-232C levels MC1489A line receiver MC1488 line driver Signal Conversion to and from TTL and RS-232C levels.

Signal meanings – RS232

Transmit data group – TD – transmit data – data from DTE to DCE – RTS – ready to send – DTE asserts before sending data and waits for CTS before sending – CTS – clear to send – DCE sends in response to RTS
• Receive data group – TD – transmit data – data from DCE to DTE – DSR – data set ready – DCE asserts before sending data and waits for DTR before sending – DTR – data terminal ready– DTE sends in response to DSR – CD – carrier detect – modem is receiving a carrier from a modem at other end
• SG – signal ground – common return for all lines

• Note: Not all manufacturers use the control lines
Connecting DTE to DCE equipment uses a straight through cable with wires connected to the proper pins. A DTE to DTE connection requires a null modem cable.
Baud: bits per second Baud Application 110 ASR-33 Teletype 300 Early acoustic modems 1200 Direct-coupled modems c. 1980 2400 Modems c. 1990 9600 Serial terminals 19200 38400 Typical maximum.

3.Write short notes about the communication protocols in detailed manner?

Communication Protocols

(1) parallel data transmission ieee-488 Parallel (HPIB or GPIB) Centronics Parallel Protocol (Printer) SCSI IDE ISA (Industrial Standard Architecture, 16 bit) PCI (Referral Component Interconnect, 32 bit) AGP

(2) serial data transmission RS 232 RS 422 RS 485 UART (Universal Asynchronous receiver-transmitter) USART (Universal Synchronous- Asynchronous receiver-transmitter) MIDI IEEE1394, also called "FireWire" CAN (Controller Area Network) USB (Universal Serial Bus) I 2 C (Inter Integrated Circuit) -- Philips SPI (Serial Peripheral Interface bus) Micro-wire Ethernet Fiber optics Bluetooth wifi sub sections to parallel and SERIAL protocols Wired connections Wireless connections Radio Frequency (RF) infrared (ir) how to selection a protocol or to develop your own Amount of data Speed of processor Hardware or software implementation Number of available pins Number of sensor Introduction to Communication Interface Parallel Communication Interface (PCI) Synchronous Asynchronous Serial Communication Interface (SCI)

Parallel ports were originally developed by IBM as a way to connect a printer to a PC. When IBM was in the process of designing the PC, the company wanted the computer to work with printers offered by Centronics, a top printer manufacturer at the time. IBM decided not to use the same port interface on the computer that Centronics used on the printer. Instead, IBM engineers coupled a 25-pin connector, DB-25, with a 36-pin Centronics connector to create a special cable to connect the printer to the computer. Other printer manufacturers ended up adopting the Centronics interface, making this strange hybrid cable an unlikely de facto standard. When a PC sends data to a printer or other device using a parallel port, it sends 8 bits of data (1 byte) at a time. These 8 bits are transmitted parallel to each other. The standard parallel port is capable of sending 50 to 100 kilobytes of data per second. Advantages of Parallel Data Transmission: ¾Fastest form of transmission -- able to send multiple bits simultaneously ¾doesn’t require high frequency of operation Disadvantages of Parallel Data Transmission: ¾Requires separate lines for each bit of a word

¾Costly to run long distances due to multiple wires ¾Suffers from electromagnetic interference ¾Cable lengths more limited than a serial cable Applications: Parallel ports can be used to connect a host of popular computer peripherals: such as prints, scanners, CD burners, external hard drives, Iomega zip, network adapters, and tape backup drives. Types of parallel port At the present time it is known four types of parallel port: o Standard parallel port (SPP) o Parallel port PS/2 (bidirectional) o Enhanced Parallel Port (EPP) o Extend Capability Port (ECP) SPP/EPP/ECP The original specification for parallel ports was unidirectional, meaning that data only traveled in one direction for each pin. With the introduction of the PS/2 in 1987, IBM offered a new bidirectional parallel port design. This mode is commonly known as Standard Parallel Port (SPP) and has completely replaced the original design. Bidirectional communication allows each device to receive data as well as transmit it. Many devices use the eight pins (2 through 9) originally designated for data. Using the sameeight pins limits communication to half-duplex, meaning that information can only travel in one direction at a time. But pins 18 through 25, originally just used as grounds, can be used as data pins also. This allows for full-duplex (both directions at the same time) communication. Enhanced Parallel Port (EPP) was created by Intel, Xircom, and Zenith in 1991. EPP allows for much more data, 500 kilobytes (KB) to 2 megabytes (MB), to be transferred each second. It was targeted specifically for non-printer devices that would attach to the parallel port, particularly storage devices that needed the highest possible transfer rate.

4.Write short notes about the serial communication in detailed manner?

Serial Data Transmission:

1. **Synchronous Data Transmission**: Data is transmitted one bit at a time, using a clock to maintain integrity between words. Advantages: ¾Only one (half duplex) or two (full duplex) wires are required to send/receive data. ¾Low cost due to low number of wires. Disadvantages: ¾Lower speeds than parallel transmissions. ¾Difficult to maintain data integrity due to problems with synchronizing clocks.

2. **Asynchronous Data Transmission**: Data is transmitted on bit at a time using start bits and strop bits to maintain integrity between words.

   Disadvantages: ¾Lower speeds than parallel transmissions. Key words for SCI Baud Rate: The measure of the number of signal elements transmitted or received per second. Baud rates and data bit rates (bps-bit per second) are not equal in

asynchronous transmission due to the start and stop bits. Start Bit: The bit preceding every word that signals the receiver a data word is coming.

In some microcontroller (e.g., HC11) the start bit is logic low (0), while in others the start bit is logic high

(1). Parity Bit: A bit sometimes added to the end of the data word. There are three possible settings for the parity: none, even, and odd. The setting represents the sum of the 1's transmitted. Stop Bit: The bit or bits following every word that signals the end of a data word. In some microcontroller (e.g., HC11) the stop bit is logic high (1), while in others the start bit is logic low (0).

Half Duplex: Two-way serial communication using only one line. With half duplex, the device can not transmit and received at the same time.

Full Duplex: Two-way serial communication using two lines. With full duplex, data can be simultaneously transmitted and received. Applications of SCI ¾The SCI can be used to transmit/receive data through a modem. ¾The SCI can be used to transmit/receive data with any device that uses RS-232-C protocol.

5.Write short notes about the serial communication standards in detailed manner?

1.RS-485

2. RS-485

3.RS-422

RS422 is an improved RS-422 with the capability to connect up to 16 devices (transceivers) on one serial bus to form a network. Such a network can have a "daisy chain" topology where each device is connected to two other devices except for the devices on the ends.

Only one device may drive data onto the bus at a time. The standard does not specify the rules for deciding who transmits and when on such a network.

That's up to the system designer to define. RS-423 RS-423 is similar to RS-232C except that it allows for higher baud rates and longer cable lengths because it tolerates ground voltage differences between sender and receiver.

The maximum signal voltage levels are ±6 volts. Ground voltage differences can occur in electrically noisy environments where heavy electrical machinery is operating. RS-422 RS-422, like RS-232, is used to connect only two systems.

It uses differential, or "double ended" data transmission, which means that data is transmitted simultaneously on two wires between two stations independent of the ground wire. Each signal requires 2 wires with a ground present in the system.

The advantage of this method over RS-232 is higher speeds and longer cable lengths - 4000 feet at a 100K baud rate, for example.

6.Write short notes about the CAN bus in detailed manner?

Controller Area Network (CAN) is a network protocol developed by Bosch for vehicle systems, but which is coming into use for linking distributed controllers, sensors etc in other fields. Bosch have published a specification .

CAN is a CSMA/CD protocol (some sources have CSMA/CR for similar protocols) that uses non-return to zero coding with bit stuffing. It supports speeds of up to 1Mb/s so is an SAE class C protocol, suitable for real time control applications.

Messages are not addressed to intended recipients, but the sender's identifier is included, and this tells the receivers what data it contains so the receiver ignores it if it is not interested. Messages are given a priority according to the sender's address, so the priority of messages is decided at the design stage.

In the spec there are two standards for CAN 2.0, imaginatively called A and B. These differ in message format , B has an extended message format, with a 29 bit identifier, as opposed to A's 11 bit one.

CAN, the CAN devices add filtration of the messages, so a controller is only interrupted by those messages the filter passes, that is those of interest to that controller.

CANbus in Automobile Electrical and Electronic Systems draw attention to the difference between having a local intelligent control module (for example, for all functions located in the driver's door) and having the intelligence actually in the actuators, so control is distributed, and each actuator (and each sensor) is on the bus itself.

Network access, collision detection and resolution

Binary zero is represented by a "dominant" state in the bus and binary one by a recessive state, so a binary zero takes precedence over a one, so lower numbered identifiers have priority over higher numbered ones.

CAN is a CSMA/CD protocol. If the network is idle, any node can send a message. If two messages are sent simultaneously, the node that sends a recessive bit, but detects a dominant bit stops transmitting, leaving the network free for the higher priority message. The higher priority message is not corrupted (Non-destructive bitwise arbitration). As this strategy

resolves collisions and does not merely detect them, some sources describe protocols with a similar collision strategy as CSMA/CR, Carrier Sense Multiple Access/Collision Resolution. The identifier and RTR fields are used for collision arbitration. Therefore arbitration breaks down if two nodes can send data (as opposed to remote request) messages with the same identifier, as the clash will not be identified until later in the message, giving rise to a bit error. Each node must send data messages with a unique identifier. This has the side effect that if, say, all four road wheels had rotation sensors would each need their own identifier, so they would have an order of priority. It seems to me not unreasonable to suggest that this could lead to conflicts in designing the system, which I do not propose to discuss here as it is outside the scope of the project.the network once the current message transmission is complete. The second highest priority message is guaranteed access after that, provided the top priority message source doesn't broadcast continuously, so this is pretty much guaranteed. Surely, however, as one moves down the order of priority, eventually one is going to reach a point where a high priority source might be ready to transmit again while a low priority source is still waiting, so its latency is not guaranteed.

7.Explain the error detection method by using CAN bus in detailed manner?

Error detection

There are 5 error detection mechanisms: -

1. Cyclic redundancy check. Each message contains a 15 bit CRC code computed by sender and checked by receivers, who will flag any errors. More in the spec (in black binder)
2. Frame check. At certain points in the frame, the correct value is predefined.
3. ACK(nowledgement) Error Check. If transmitter determines an error has not been acknowledged, an ACK error is flagged.
4. Bit Monitoring. A transmitter checks the network and flags a bit error if the value on the bus is not that sent. This does not happen during transmission of the identifier field, of course, as that is how a collision is detected.
5. Bit stuffing After 5 consecutive bits of the same value, a bit of the opposite value is added to the frame.

If an error is detected, an error frame is sent, aborting the transmission.

Error confinement provides a mechanism for distinguishing between temporary and permanent errors. Each node has two error counters (for transmit and receive) which are

incremented when errors are found. It is covered in more detail in the spec but briefly each receive error increments its counter by one, and each transmit error increments its counter by 8. If either counter goes above 127 the node concerned goes into “error passive” mode. In this mode it can still transmit and receive messages, but is restricted in flagging errors. If a device’s transmit error counter goes above 255, the device will go into “bus off” mode and will cease to be active. This condition will clearly need to be modelled in simulating CANbus systems for FMEA. This seems to imply that we must allow for the modelling of repeat errors or for modelling the network as though the counter(s) had reached a level such that devices were going into “bus off” mode.

Bit timing and synchronisation

This is covered in the spec of course, and there is an introduction to this in the Omegas material . Briefly, a bit time consists of four non-overlapping segments, Sync-seg, Prop-seg, Phase-seg1 and Phase-seg2. An edge should lie within Sync-seg, while Prop-seg is used to compensate for delay times in the network. It is therefore the sum of twice the signal propagation time on the bus, the input comparator delay and the output driver delay, so is characteristic to the network. Phase-seg1 and Phase-seg2 are used to compensate for edge phase errors. They can be lengthened or shortened by resynchronisation. The sampling point is the boundary between Phase-seg1 and Phase-seg2. As non-return to zero encoding is used, there need not be an edge during Sync-seg, but bit stuffing ensures that there will be an edge after five edge-free ones.

CAN in the ISO/OSI stack and higher level protocols

The scope of the CANbus protocol covers the physical and data link layers of the ISO/OSI model. to three levels in the CANbus protocol; physical layer, transfer layer and object layer. The physical layer is not defined in the Bosch spec, but is typically shielded or unshielded twisted pair. Idle state is both lines at +2.5 volts. A dominant bit reduces one line, known as CAN_L, to zero, while increasing the other line (CAN_H) to +5 volts while a recessive bit is close to the idle value, with CAN_L slightly above CAN_H, so is “over written” by a dominant bit. A standard for the physical layer of a 500 KBPS vehicle network is defined in SAE J2284-500 .The transfer and object layers between them comprise all the services and functions of the ISO/OSI data link layer..

The Can in Automation (CiA) trade organisation supports various higher level protocols

- CANopen
- DeviceNet
- CAL (CAN application layer)
- CAN Kingdom
- SDS (Smart Distributed System)

CiA is an organisation mainly interested in using CAN for industrial automation so it may well be that the protocols listed above are more common in that field than in the automotive field.

8.Explain the input output port types in detailed manner?

UART protocol serial line format

Starting point of receiving the bits for each byte is indicated by a line transition from 1 to 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

UART protocol serial line format

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

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

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

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 datainput 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 for the serial line bits is (1/T) bps. [1bps = 1 bit per s]

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)

9.Explain the serial peripheral interface (SPI) in detailed manner?

SPI = Simple, 3 wire, full duplex, synchronous serial data transfer Interfaces to many devices, even many non-SPI peripherals Can be a master or slave interface

4 interface pins: -

MOSI master out slave in

MISO master in slave out

SCK serial clock

SS_n slave select 3 registers:

1.SPCR control register

2.SPSR status register

3.SPDR data register Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) SPI “Gotchas” “Now my board won’t program.” SPI shares SCK with programming interface. If it won’t program anymore, you likely messed up SCK. “SPI acts totally wierd.” Often a symptom of SS_n being configured as an input and being left to float or allowed to go high.

SPI goes in and out between slave and master modes"SPI device interactions:" When programming, the programmer first does a chip reset. When the mega128 resets, all pins are set to input with high impedance (floating). If a SPI device is on the SPI bus, its chip-select may float low and enable the device, and SPI data will crash the programming data. Adding a pull-up resistor to chip selects will solve this problem.

the SPI clock to run, a "dummy" write is made to the SPI SPDR register. This starts the clock running so the data on MISO is brought into the uC. If no peripherals are selected, the outgoing data will be ignored. If you are clever, you can send data out and bring data in at the same time.

spi_read //Reads the SPI port.

uint8_t spi_read(void){ SPDR = 0x00;

//"dummy" write to SPDR while (bit_is_clear(SPSR,SPIF)){}

//wait till 8 clock cycles are done return(SPDR);

//return incoming data from SPDR }//read_spi

Serial Peripheral Interface (SPI) SPI Application - Code

spi_init //Initializes the SPI port on the mega128. Does not do any further //

external device specific initializations.

/ void spi_init(void){ DDRB = 0x07;

//Turn on SS, MOSI, SCLK (SS is output)

10.Explain the device driver in detailed manner?

Block and character devices

- Block devices include disk drives
- Commands include read, write, seek
- Raw I/O or file-system access
- Memory-mapped file access possible
- Character devices include keyboards, mice, serial ports
- Commands include get, put
- Libraries layered on top allow line editing

Network devices

- Different enough from the block & character devices to have own interface
- Unix and Windows/NT include socket interface – Separates network protocol from network operation
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes

Clocks and timers

- Provide current time, elapsed time, timer
- if programmable interval time used for timings, periodic interrupts
- ioctl (on UNIX) covers odd aspects of I/O such as clocks and timers Blocking and nonblocking I/O
- Blocking - process suspended until I/O completed – Easy to use and understand – Insufficient for some needs
- Nonblocking - I/O call returns as much as available – User interface, data copy (buffered I/O) – Implemented via multi-threading – Returns quickly with count of bytes read or written
- Asynchronous - process runs while I/O executes – Difficult to use – I/O subsystem signals process when I/O completed

Device driver design issues

- Operating system and driver communication
- Commands and data between OS and device drivers
- Driver and hardware communication – Commands and data between driver and hardware
- Driver operations – Initialize devices – Interpreting commands from OS – Schedule multiple outstanding requests – Manage data transfers – Accept and process interrupts – Maintain the integrity of driver and kernel data structures

Device driver interface

- Open(deviceNumber) – Initialization and allocate resources (buffers)
- Close(deviceNumber) – Cleanup, deallocate, and possibly turnoff
- Device driver types – Block: fixed sized block data transfer – Character: variable sized data transfer – Terminal: character driver with terminal control – Network: streams for networking

UNIT III

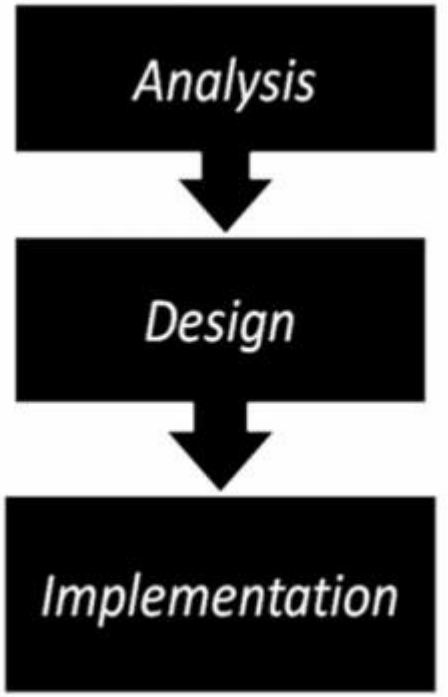
1.Write the short notes about the EDLC in detail manner?

EMBEDDED PRODUCT DEVELOPMENT LIFE CYCLE (EDLC)

EDLC is Embedded Product Development Life Cycle

It is an Analysis – Design – Implementation based problem solving approach for embedded systems development.

There are three phases to Product development:



Analysis involves understanding what product needs to be developed

- Design involves what approach to be used to build the product
- Implementation is developing the product by realizing the design.

Need for EDLC

- EDLC is essential for understanding the scope and complexity of the work involved in embedded systems development
- It can be used in any developing any embedded product
- EDLC defines the interaction and activities among various groups of a product development phase.
- Example:-project management, system design

Objectives of EDLC

- The ultimate aim of any embedded product in a commercial production setup is to produce Marginal benefit
- Marginal is usually expressed in terms of Return On Investment
- The investment for product development includes initial investment, manpower, infrastructure investment etc.
- EDLC has three primary objectives are:

Ensure that high quality products are delivered to user

1. Quality in any product development is Return On Investment achieved by the product
2. The expenses incurred for developing the product the product are:-
3. Initial investment
4. Developer recruiting
5. Training
6. Infrastructure requirement related

Risk minimization defect prevention in product development through project management

i. In which required for product development 'loose' or 'tight' project management
ii. 'project management is essential for ' predictability co-ordination and risk minimization
iii. Resource allocation is critical and it is having a direct impact on investment
iv. Example:- Microsoft @ Project Tool

Maximize the productivity

i. Productivity is a measure of efficiency as well as Return On Investment
ii. This productivity measurement is based on total manpower efficiency
iii. Productivity in which when product is increased then investment is fall down,Saving manpower

2.Write the short notes about the different phases of EDLC in detail manner?

different phases of edlc

The following figure depicts the different phases in EDLC:

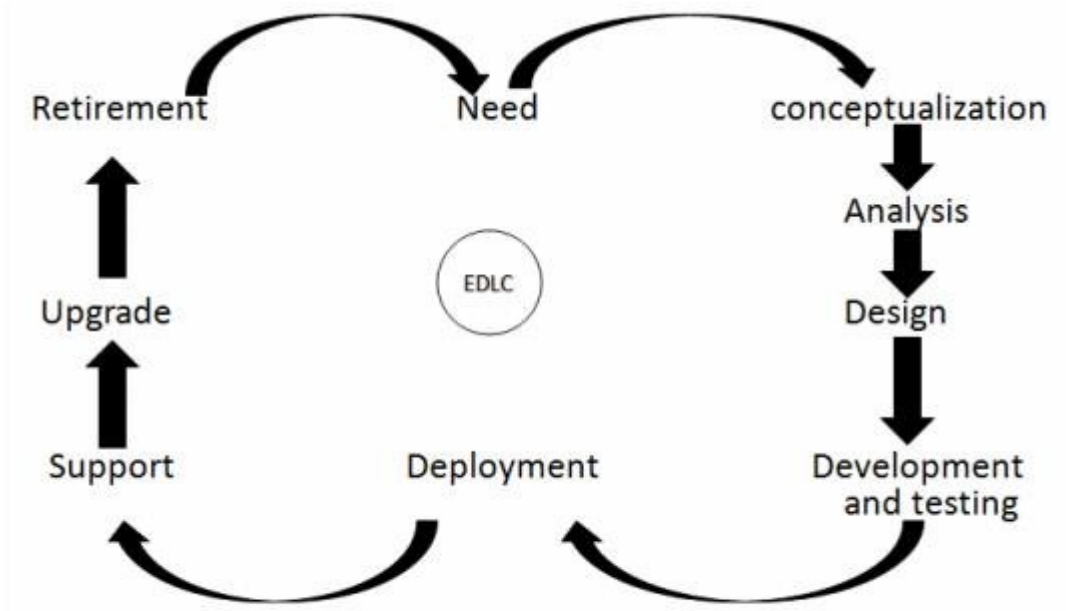


Figure : Phases of EDLC

Need

- The need may come from an individual or from the public or from a company.
- 'Need' should be articulated to initiate the Development Life Cycle; a 'Concept Proposal' is prepared which is reviewed by the senior management for approval.

Need can be visualized in any one of the following three needs:

- New or Custom Product Development.
- Product Re-engineering.
- Product Maintenance.

Conceptualization

Defines the scope of concept, performs cost benefit analysis and feasibility study and prepare project management and risk management plans.

The following activities performed during this phase:

Feasibility Study : Examine the need and suggest possible solutions.

Cost Benefit Analysis (CBA): Revealing and assessing the total development cost and profit expected from the product.

Product Scope: Deals with the activities involved in the product to be made.

Planning Activities: Requires various plans to be developed first before development like Resource Planning & Risk management Plans.

Analysis

The product is defined in detail with respect to the inputs, processes, outputs, and interfaces at a functional level.

The various activities performed during this phase..

• **Analysis and Documentations:** This activity consolidates the business needs of the product under development.

• **Requirements that need to be addressed..**

Functional Capabilities like performance θ

Operational and non-operational quality attribute θ

Product external interface requirements θ

User manuals θ Data requirements θ

Operational requirements θ

Maintenance requirements θ

General assumptions θ

Defining Test Plan and Procedures: The various type of testing performed in a product development are:

Unit testing – Testing Individual modules

Integration testing – Testing a group of modules for required functionality

System testing- Testing functional aspects or functional requirements of the product after integration

User acceptance testing- Testing the product to meet the end user requirements.

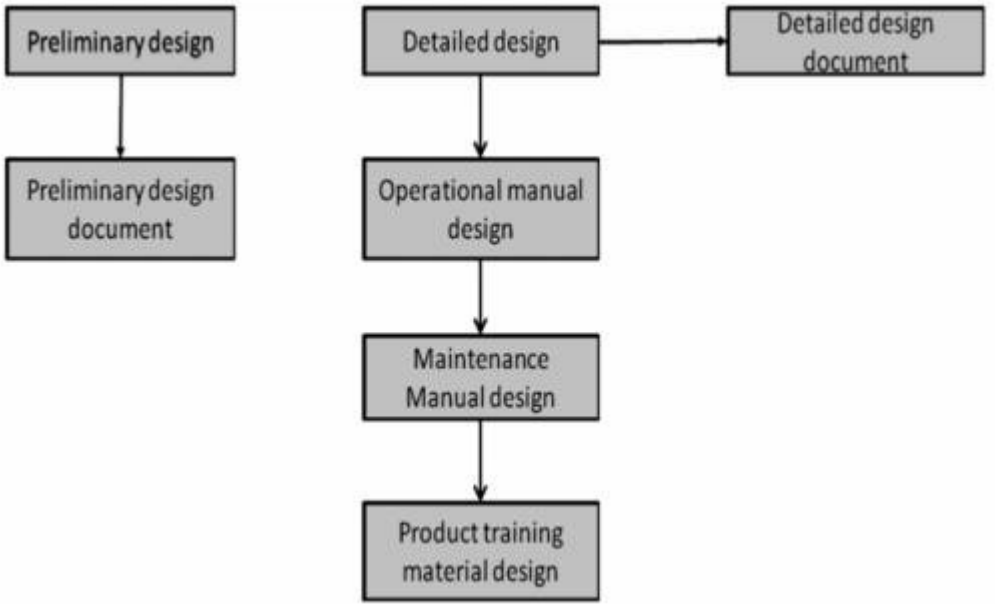
Design

The design phase identifies application environment and creates an overall architecture for the product.

It starts with the Preliminary Design. It establishes the top level architecture for the product. On completion it resembles a 'black box' that defines only the inputs and outputs. The final product is called Preliminary Design Document (PDD).

Once the PDD is accepted by the End User the next task is to create the 'Detailed Design'.

It encompasses the Operations manual design, Maintenance Manual Design and Product Training material Design and is together called the 'Detailed Design Document'.



3.Write the short notes about the testing and development procedure of EDLC in detail manner?

Development and Testing

Development phase transforms the design into a realizable product.

The detailed specification generated during the design phase is translated into hardware and firmware.

The Testing phase can be divided into independent testing of firmware and hardware that is:

- Unit testing
- Integration testing
- System testing
- User acceptance testing

Deployment

Deployment is the process of launching the first fully functional model of the product in the market.

It is also known as First Customer Shipping (FCS).

Tasks performed during this phase are:

Notification of Product Deployment: Tasks performed here include:

Deployment schedule

- Brief description about the product
- Targeted end user
- Extra features supported
- Product support information

Execution of training plan

Proper training should be given to the end user top get them acquainted with the new product.

Product installation

Install the product as per the installation document to ensure that it is fully functional.

Product post Implementation Review

After the product launch, a post implementation review is done to test the success of the product.

Support

- The support phase deals with the operational and maintenance of the product in the production environment.
- Bugs in the product may be observed and reported.
- The support phase ensures that the product meets the user needs and it continues functioning in the production environment.
- Activities involved under support are

Setting up of a dedicated support wing: Involves providing 24 x 7 supports for the product after it is launched.

Identify Bugs and Areas of Improvement: Identify bugs and take measures to eliminate them.

Upgrades

- Deals with the development of upgrades (new versions) for the product which is already present in the market.
- Product upgrade results as an output of major bug fixes.
- During the upgrade phase the system is subject to design modification to fix the major bugs reported.

Retirement/Disposal

- The retirement/disposal of the product is a gradual process.
- This phase is the final phase in a product development life cycle where the product is declared as discontinued from the market.
- The disposal of a product is essential due to the following reasons
- Rapid technology advancement
- Increased user needs

4 eldc approaches

Following are some of the different types of approaches that can be used to model embedded products.

1. Waterfall or Linear Model
2. Iterative/ Incremental or Fountain Model
3. Prototyping Model
4. Spiral Model

4.Describe the types of models in EDLC in detail manner?

1 Introduction

2 Waterfall or Linear Model

3 Iterative/ Incremental or Fountain Model

4 Prototyping Model

5 Spiral Model

OBJECTIVES

After reading this chapter you will understand:

Some EDLC Models like:

- Waterfall or Linear Model
- Iterative/ Incremental or Fountain Model
- Prototyping Model,Spiral Model

embedded development life cycle

Objectives

1 Introduction

2 EDLC

- Need For ELDC
- Objectives

3 Different Phases of EDLC

4 ELDC Approaches

OBJECTIVES

- After Reading this chapter you will understand
- The Embedded Development Life Cycle
- Phases Involved in the EDLC

The SDLC used in Software Development, there is EDLC used in Embedded product development. This chapter explains what is the EDLC, its objectives, the phases that are involved in the EDLC.

5.Describe the waterfall models in detail manner?

waterfall model

- Linear or waterfall model is the one adopted in most of the olden systems.
- In this approach each phase of EDLC (Embedded Development Product Lifecycle) is executed in sequence.
- It establishes analysis and design with highly structured development phases.
- The execution flow is unidirectional.
- The output of one phase serves as the input of the next phase
- All activities involved in each phase are well planned so that what should be done in the next phase and how it can be done.
- The feedback of each phase is available only after they are executed.
- It implements extensive review systems To ensure the process flow is going in the right direction.
- One significant feature of this model is that even if you identify bugs in the current design the development process proceeds with the design.
- The fixes for the bug are postponed till the support phase.

Advantages

Product development is rich in terms of:

- Documentation
- Easy project management
- Good control over cost & Schedule

Drawbacks

It assumes all the analysis can be done without doing any design or implementation

- The risk analysis is performed only once.
- The working product is available only at the end of the development phase
- Bug fixes and correction are performed only at the maintenance/support phase of the life cycle.

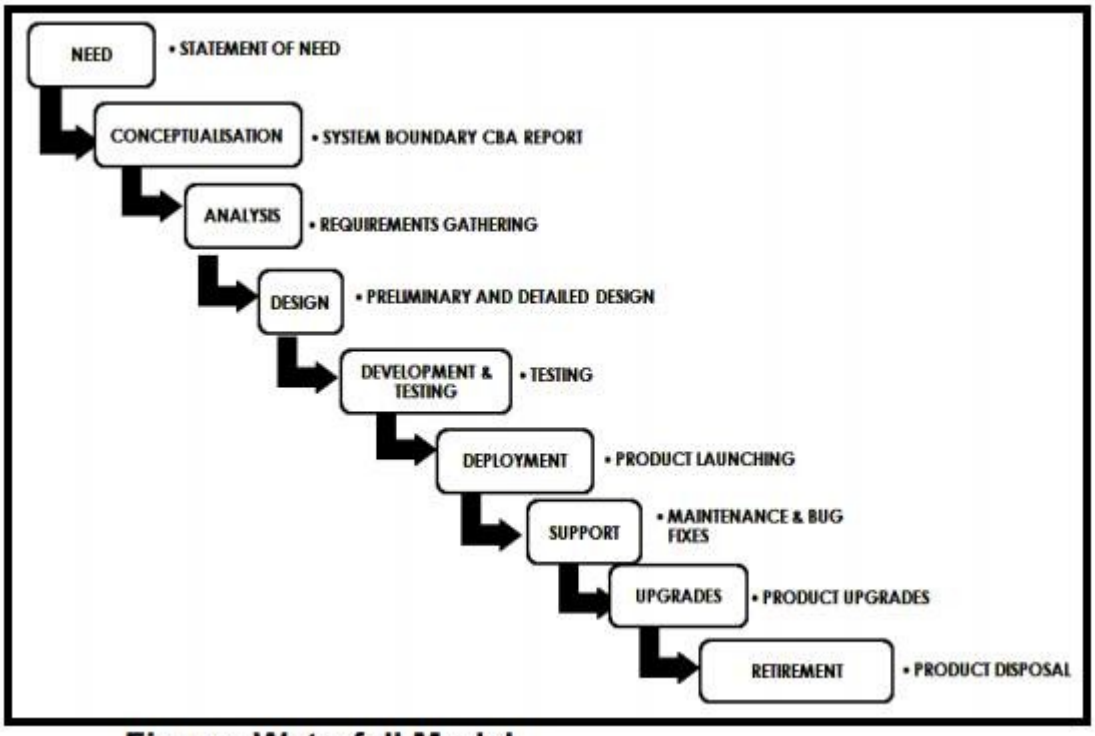


Figure: Waterfall Model

6.Describe the iterative models of EDLC in detail manner?

Iterative / incremental or fountain model

- Iterative and Incremental development is at the heart of a cyclic software development process developed in response to the weaknesses of the waterfall model.
- The iterative model is the repetitive process in which the Waterfall model is repeated over and over to correct the ambiguities observed in each iteration.

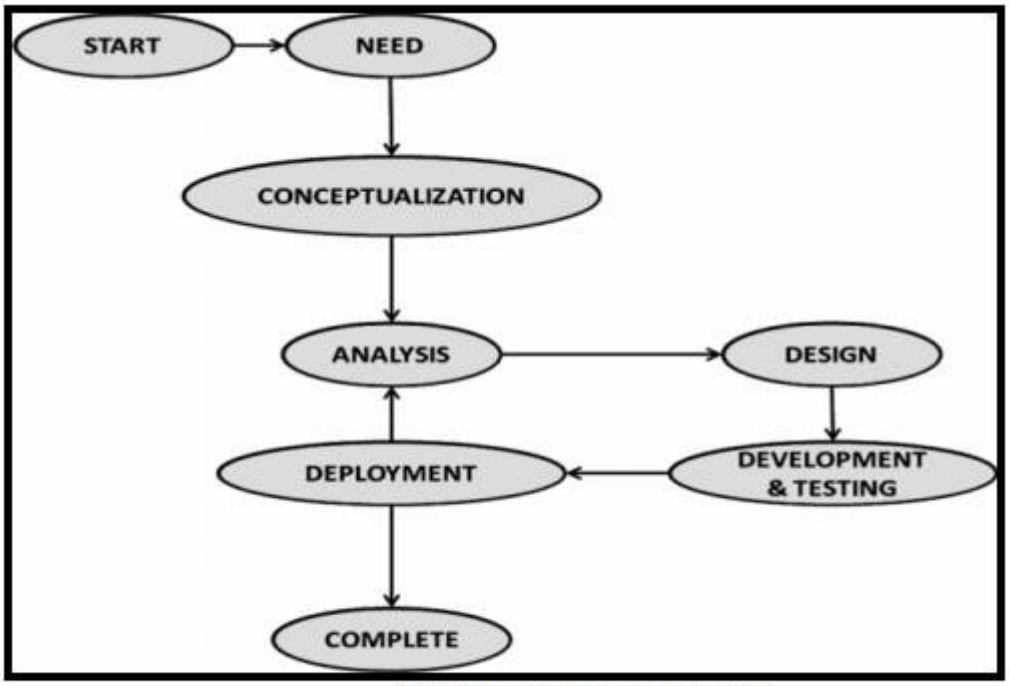


Figure: Iterative model

The above figure illustrates the repetitive nature of the Iterative model.

- The core set of functions for each group is identified in the first cycle, it is then built, deployed and release. This release is called as the first release.
- Bug fixes and modification for first cycle carried out in second cycle.
- Process is repeated until all functionalities are implemented meeting the requirements.

Advantages

- Good development cycle feedback at each function/feature implementation
- Data can be used as reference for similar product development in future.
- More responsive to changing user needs.
- Provides working product model with at least minimum features at the first cycle.
- Minimized Risk
- Project management and testing is much simpler compared to linear model.
- Product development can be stopped at any stage with a bare minimum working product.

Disadvantages

- Extensive review requirement each cycle.
- Impact on operations due to new releases.
- Training requirement for each new deployment at the end of each development cycle.
- Structured and well documented interface definition across modules to accommodate changes

7.Describe the prototyping models of EDLC in detail manner?

prototyping model

- It is similar to iterative model and the product is developed in multiple cycles.
- The only difference is that, Prototyping model produces a refined prototype of the product at the end of each cycle instead of functionality/feature addition in each cycle as performed by the iterative model.
- There won't be any commercial deployment of the prototype of the product at each cycle's end.
- The shortcomings of the proto-model after each cycle are evaluated and it is fixed in the next cycle.
- After the initial requirement analysis, the design for the first prototype is made, the development process is started.
- On finishing the prototype, it is sent to the customer for evaluation.
- The customer evaluates the product for the set of requirements and gives his/her feedback to the developer in terms of shortcomings and improvements needed.
- The developer refines the product according to the customer's exact expectation and repeats the proto development process.
- After a finite number of iterations, the final product is delivered to the customer and launches in the market/operational environment
- the product undergoes significant evolution as a result of periodic shuttling of product information between the customer and developer

The prototyping model follows the approach

- Requirement definition
- Proto-type development
- Proto-type evaluation
- Requirements refining

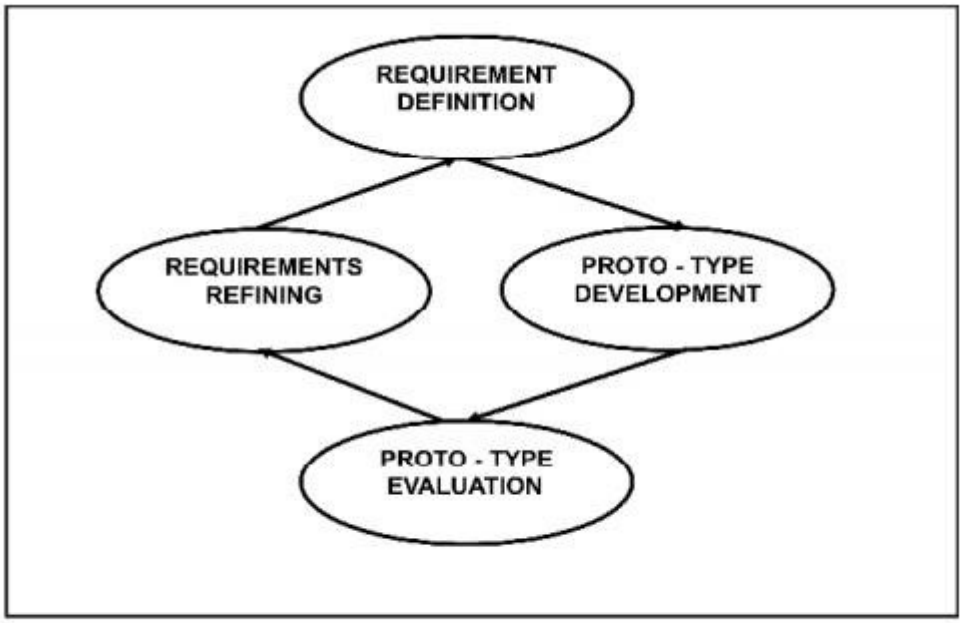


Figure: Prototyping Model

8.Describe the spiral models of EDLC in detail manner?

spiral model

- Spiral model is developed by Barry Boehm in 1988.
- The Product development starts with project definition and traverse through all phases of EDLC(Embedded Product Development Life Cycle).

The activities involved are:

- Determine objectives, alternatives, constraints
- Evaluate alternatives, identify and resolve risks

It is a combines the concept of Linear Model and iterative nature of Prototyping Model.

- In prototyping after the requirement analysis the design for the prototype is made and development process is started.
- On finishing the prototype it is send to the customer for evaluation ie. Judgment.
- After customer evaluation for the product the feedback is taken from the customer in term of what improvement is needed.
- Then developer refines the product according to the customer expectation.

Spiral Model contains the concept of linear model, having following type.

- Requirement
- Analysis
- Design
- Implementation

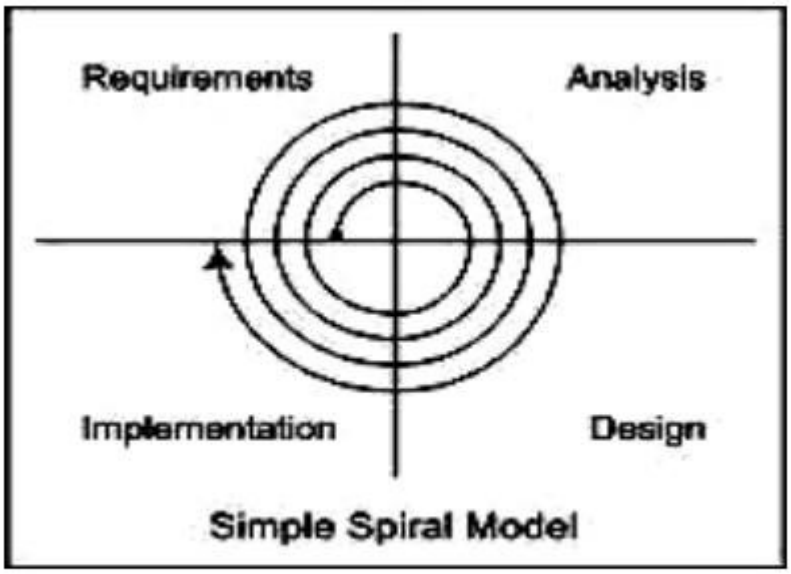


Figure: Spiral Model

Requirement:

- This process is focused specifically on embedded software, to understand the nature of the software to be build and what are the requirement for the software.And the requirement for both the system & the software is documented & viewed to customer.

Analysis:

- Analysis is performed to develop a detailed functional module under consideration.
- The product is defined in detailed with respect to the input, processing & output.
- This phase emphasis on determining 'what function must be performed by the product' & how to perform those function.

Design:

- Product design deals with the entire design of the product taking the requirement into consideration.
- The design phase translates requirement into representation**.**

Implementation:

- The launching of first fully functional model of the product in the market is done or handing over the model to an end user/client
- modifications are implemented & product is made operational in production environment.

9.Explain the different types of trends in embedded systems?

Objectives

1 Introduction

2 Processor Trends

3 Operating System Trends

4 Development Language Trends

5 Open Standards, Frameworks and alliances

6 Bottlenecks faced by Embedded Industry

Different trends in the embedded industry related to:

- Processor Trends
- Operating System Trends
- Development Language Trends
- Open Standards, Frameworks and alliances
- Bottlenecks faced by Embedded Industry

processor trends

There have been tremendous advancements in the area of processor design.

Following are some of the points of difference between the first generation of processor/controller and today's processor/ controller.

Number of ICs per chip: Early processors had a few number of IC/gates per chip. Today's processors with Very Large Scale Integration (VLSI) technology can pack together ten of thousands of IC/gates per processor.

Need for individual components: Early processors need different components like brown out circuit, timers, DAC/ADC separately interfaced if required to be used in the circuit. Today's processors have all these components on the same chip as the processor.

Speed of Execution: Early processors were slow in terms of number of instructions executed per second. Today's processor with advanced architecture support features like instruction pipeline improving the execution speed.

Clock frequency: Early processors could execute at a frequency of a few MHz only. Today's processors are capable of achieving execution frequency in rage of GHz.

Application specific processor: Early systems were designed using the processors available at that time. Today it is possible to custom create a processor according to a product requirement.

Following are the major trends in processor architecture in embedded development.

System on Chip (SoC)

- This concept makes it possible to integrate almost all functional systems required to build an embedded product into a single chip.
- SoC are now available for a wide variety of diverse applications like Set Top boxes, Media Players, PDA, etc.
- SoC integrate multiple functional components on the same chip thereby saving board space which helps to miniaturize the overall design.

Multicore Processors/ Chiplevel Multi Processor

This concept employs multiple cores on the same processor chip operating at the same clock frequency and battery.

Based on the number of cores, these processors are known as:

Dual Core – 2 cores

Tri Core – 3 cores

Quad Core – 4 cores

These processors implement multiprocessing concept where each core implements pipelining and multithreading.

Reconfigurable Processors

- It is a processor with reconfigurable hardware features.
- Depending on the requirement, reconfigurable processors can change their functionality to adapt to the new requirement. Example: A reconfigurable processor chip can be configured as the heart of a camera or that of media player.
- These processors contain an Array of Programming Elements (PE) along with a microprocessor. The PE can be used as a computational engine like ALU or a memory element.

operating system trends

- The advancements in processor technology have caused a major change in the Embedded Operating System Industry.

- There are lots of options for embedded operating system to select from which can be both commercial and proprietary or Open Source.
- Virtualization concept is brought in picture in the embedded OS industry which replaces the monolithic architecture with the microkernel architecture.
- This enables only essential services to be contained in the kernel and the rest are installed as services in the user space as is done in Mobile phones.
- Off the shelf OS customized for specific device requirements are now becoming a major trend.

development language trends

There are two aspects to Development Languages with respect to Embedded Systems Development

Embedded Firmware

- It is the application that is responsible for execution of embedded system.
- It is the software that performs low level hardware interaction, memory management etc on the embedded system.

Embedded Software

- It is the software that runs on the host computer and is responsible for interfacing with the embedded system.
- It is the user application that executes on top of the embedded system on a host computer.
- Early languages available for embedded systems development were limited to C & C++ only. Now languages like Microsoft C$, ASP.NET, VB, Java, etc are available.

10.Explain the design process of embedded systems in detailed manner?

The embedded system design process

Overview of the embedded system design process aimed at two objectives. First,it will give us an introduction to the various steps in embedded system design before we delve into them in more detail. Second, it will allow us to consider the design ***methodology*** itself. summarizes the major steps in the embedded system design process.In this top–down view,we start with the system ***requirements*** in the next step comes ***Specification.***

Top–down Design—we will begin with the most abstract description of the system and conclude with concrete details. The alternative is a **bottom–up** view in which we start with components to build a system. Bottom–up design steps are shown in the figure as dashed-line arrows. We need bottom–up design because we do not have perfect insight into how later stages of the design process will turn out. During the design process we have to consider the major goals of the design such as

- manufacturing cost;
- performance (both overall speed and deadlines); and
- power consumption.

Design steps are detailed below:

A.Requirements:

Requirements may be ***functional*** or ***nonfunctional***. We must capture the basic functions of the embedded system, but functional description is often not sufficient. Typical nonfunctional requirements include:

1. *Performance**:*** The speed of the system is often a major consideration both for the usability of the system and for its ultimate cost. Performance may be a combination of soft performance metrics such as approximate time to perform a user-level function and hard deadlines by which a particular operation must be completed.

2. *Cost:* The target cost or purchase price for the system is almost always a consideration. Cost typically has two major components:

- ***Manufacturing cost*** includes the cost of components and assembly;
- ***NonRecurring engineering*** **(NRE)** costs include the personnel and other costs of designing the system.

3. *Physical size and weight:* The physical aspects of the final system can vary greatly depending upon the application. e.g) An industrial control system for an assembly line may be designed to fit into a standard-size rack with no strict limitations on weight. But a handheld device typically has tight requirements on both size and weight that can ripple through the entire system design.

Power consumption: Power, of course, is important in battery-powered systems and is often important in other applications as well. Power can be specified in the requirements stage in terms of battery life.

mock-up. The mock-up may use canned data to simulate functionality in a restricted demonstration, and it may be executed on a PC or a workstation. But it should give the customer a good idea of how the system will user can react to it. Physical,nonfunctional models of devices can also give customers a better idea of characteristics such as size and weight.

e.g) *Requirements analysis of a GPS moving map*

The moving map is a handheld device that displays for the user a map of the terrain around the user's current position; the map display changes as the user and the map device change position.

11.Explain the architecture of emdedded design process in detailed manner?

Architecture Design

- The specification does not say how the system does things, only what the system does. Describing how the system implements those functions is the purpose of the architecture.
- A sample system architecture in the form of a block diagram that shows major operations and data flows among them.
- Many implementation details should we refine that system block diagram into two block diagrams: one for hardware and another for software. These two more refined block.
- The hardware block diagram clearly shows that we have one central CPU surrounded by memory and I/O devices.
- In particular, we have chosen to use two memories: a frame buffer for the pixels to be displayed and a separate program/data memory for general use by the CPU .
- Functionality: This system is designed for highway driving and similar uses, not nautical or aviation uses that require more specialized databases and functions. The system should show major roads and other landmarks available in standard topographic databases.
- User interface*:* The screen should have at least 400_600 pixel resolution. The device should be controlled by no more than three buttons.
- Performance: The map should scroll smoothly. Upon power-up, a display should take no more than one second to appear, and the system should be able to verify its position and display the current map within 15 s.
- Cost: The selling cost (street price) of the unit should be no more than $100.
- Physical size and weight*:* The device should fit comfortably in the palm of the hand.
- Power consumption*:* The device should run for at least eight hours on four AA batteries.

Designing Hardware and Software Components

The component design effort builds those components in conformance to the architecture and specification. The components will in general include both hardware—FPGAs, boards, and so on—and software modules. Some of the components will be ready-made. In the moving map, the GPS receiver is a good example of a specialized component that will nonetheless be a predesigned, standard component. We can also make use of standard software modules. One good example is the topographic database.

System Integration

The components built are put together and see how the system works. If we debug only a few modules at a time, we are more likely to uncover the simple bugs and able to easily recognize

them. Only by fixing the simple bugs early will we be able to uncover the more complex or obscure bugs.

UNIT IV

1.Explain the memory devices in detailed manner?

we introduce the basic types of memory components that are commonly used in embedded systems. Now that we understand the operation of the bus, we are able to understand the pinouts of these memories and how values are read and written. We also need to understand the varieties of memory cells that are used to build memories. There are several varieties of both read-only and read/write memories, each with its own advantages. After discussing some basic characteristics of memories, we describe RAMs and then ROMs.

Memory Device Organization

The most basic way to characterize a memory is by its capacity, such as 256 MB. However, manufacturers usually make several versions of a memory of a given size, each with a different data width. For example, a 256-MB memory may be available in two versions:

As a 64 M 4-bit array, a single memory access obtains an 8-bit data item, with a maximum of 2^{26} different addresses.

As a 32 M 8-bit array,a single memory access obtains a 1-bit data item, with a maximum of 2^{23} different addresses.

The height/width ratio of a memory is known as its ***aspect ratio***. The best aspect ratio depends on the amount of memory required. Internally, the data are stored in a two-dimensional array of memory cells as shown in Figure 4.15. Because the array is stored in two dimensions,the *n*-bit address received by the chip is split into a row and a column address (with *n r c*).

RAS and CAS can therefore become valid at the same time. The address lines are not shown in full detail here; some address lines may not be active depend- ing on the mode in use. SDRAMs use a separate refresh signal to control refreshing. DRAM has to be refreshed roughly once per millisecond. Rather than refresh the entire memory at once, DRAMs refresh part of the memory at a time. When a section of memory is being refreshed, it cannot be accessed until the refresh is complete. The memory refresh occurs over fairly few seconds so that each section is refreshed every few microseconds.

SDRAMs include registers that control the mode in which the SDRAM operates. SDRAMs support burst modes that allow several sequential addresses to be accessed by

sending only one address. SDRAMs generally also support an interleaved mode that exchanges pairs of bytes.

Read-Only Memories

Read-only memories ***(ROMs)*** are preprogrammed with fixed data. They are very useful in embedded systems since a great deal of the code, and perhaps some data, does not change over time. Read-only memories are also less sensitive to radiation- induced errors.

There are several varieties of ROM available. The first-level distinction to be made is between **factory-programmed ROM** (sometimes called **mask-programmed ROM**) and **field-programmable ROM** . Factory-programmed ROMs are ordered from the factory with particular programming. ROMs can typically be ordered in lots of a few thousand, but clearly factory programming is useful only when the ROMs are to be installed in some quantity.

Field-programmable ROMs, on the other hand, can be programmed in the lab. **Flash memory** is the dominant form of field-programmable ROM and is electrically erasable. Flash memory uses standard system voltage for erasing and programming.

2.Explain the input output devices in detailed manner?

i/o devices

some input and output devices commonly used in embed-ded computing systems. Some of these devices are often found as on-chip devices in micro-controllers; others are generally implemented separately but are still commonly used. Looking at a few important devices now will help us understand both the requirements of device interfacing in this chapter and the uses of devices in programming in this and later chapters.

Timers and Counters

Timers and ***counters*** are distinguished from one another largely by their use, not their logic. Both are built from adder logic with registers to hold the current value, with an increment input that adds one to the current register value. However, a timer has its count connected to a periodic clock signal to measure time intervals, while a counter has its count input connected to an aperiodic signal in order to count the number of occurrences of some external event. Because the same logic can be used for either purpose, the device is often called a ***counter/timer*** .

an *n*-bit register to store the current state of the count and an array of *half subtractors* to decrement the count when the count signal is asserted. Combinational logic checks when the count equals zero; the *done* output signals the zero count. It is often useful to be able to control the time-out, rather than require exactly 2n events to occur. For this purpose, a reset register provides the value with which the count register is to be loaded. The counter/timer provides logic to load the reset register. Most counters provide both cyclic and acyclic modes of operation. In the cyclic mode, once the counter reaches the done state, it

is automatically reloaded and the counting process continues. In acyclic mode, the counter/timer waits for an explicit signal from the microprocessor to resume counting.

A ***watchdog timer*** is an I/O device that is used for internal operation of a system. As shown in Figure 4.18, the watchdog timer is connected into the CPU bus and also to the CPU's reset line. The CPU's software is designed to periodically reset the watchdog timer, before the timer ever reaches its time-out limit. If the watchdog timer ever does reach that limit, its time-out action is to reset the processor. In that case, the presumption is that either a software flaw or hardware problem has caused the CPU to misbehave. Rather than diagnose the problem, the system is reset to get it operational as quickly as possible.

Keyboards

A keyboard is basically an array of switches, but it may include some internal logic to help simplify the interface to the microprocessor. In this chapter, we build our understanding from a single switch to a microprocessor-controlled keyboard.

A switch uses a mechanical contact to make or break an electrical circuit. The major problem with mechanical switches is that they bounce as shown in Figure 4.19. When the switch is depressed by pressing on the button attached to the switch's arm, the force of the depression causes the contacts to bounce several times until they settle down. If this is not corrected, it will appear that the switch has been pressed several times, giving false inputs. A hardware debouncing circuit can be built using a one-shot timer. Software can also be used to debounce switch inputs. A raw keyboard can be assembled from several switches. Each switch in a raw keyboard has its own pair of terminals, making raw keyboards impractical when a large number of keys is required.

More expensive keyboards, such as those used in PCs, actually contain a microprocessor to preprocess button inputs. PC keyboards typically use a 4-bit microprocessor to provide the interface between the keys and the computer. The microprocessor can provide debouncing, but it also provides other functions as well. An encoded keyboard uses some code to represent which switch is cur- rently being depressed. At the heart of the encoded keyboard is the scanned array of switches shown in Figure 4.20. Unlike a raw keyboard, the scanned keyboard array reads only one row of switches at a time. The demultiplexer at the left side of the array selects the row to be read. When the scan input is 1, that value is trans- mitted to one terminal of each key in the row. If the switch is depressed, the 1 is sensed at that switch's column. Since only one switch in the column is activated, that value uniquely identifies a key. The row address and column output can be used for encoding, or circuitry can be used to give a different encoding.

3.Explain how the component interfaching was done with devices?

component interfacing

Building the logic to interface a device to a bus is not too difficult but does take some attention to detail. We first consider interfacing memory components to the bus, since that is relatively simple, and then use those concepts to interface to other types of devices.

Memory Interfacing

a memory of the exact size we need, then the memory structure is simple. If we need more memory than we can buy in a single chip, then we must construct the memory out of several chips. We may also want to build a memory that is wider than we can buy on a single chip; for example, we cannot generally buy a 32- bit-wide memory chip. We can easily construct a memory of a given width (32 bits, 64 bits, etc.) by placing RAMs in parallel.

logic to turn the bus signals into the appropriate memory signals. For example, most busses won't send address signals in row and column form. We also need to generate the appropriate refresh signals.

Device Interfacing

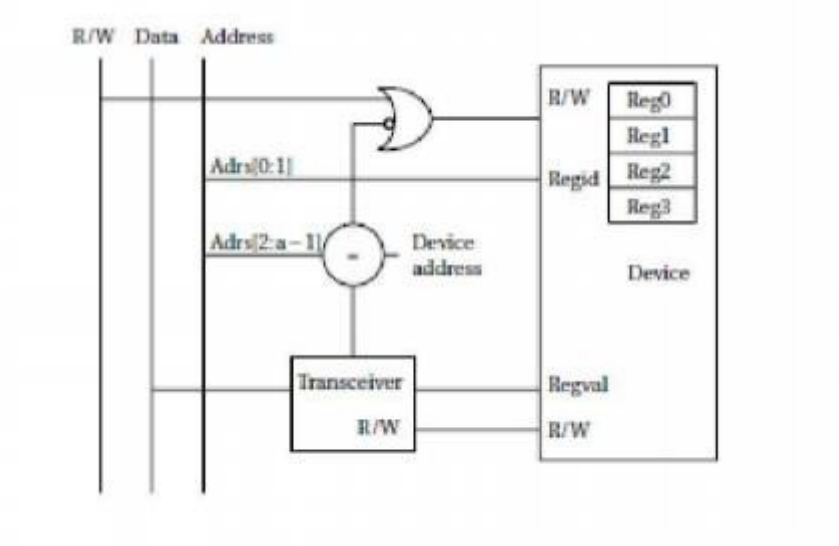
Some I/O devices are designed to interface directly to a particular bus, forming glueless interfaces. But glue logic is required when a device is connected to a bus for which it is not designed.

An I/O device typically requires a much smaller range of addresses than a memory, so addresses must be decoded much more finely. Some additional logic is required to cause the bus to read and write the device's registers. Example 4.1 shows one style of interface logic.

Example

A glue logic interface

Below is an interfacing scheme for a simple I/O device



The device has four registers that can be read and written by presenting the register number on the *regid*pins, asserting R/W as required, and reading or writing the value on the *regval* pins.

4.Explain the interrupts routine of RTOS in detailed manner?

interrupts

Busy-wait I/O is extremely inefficient—the CPU does nothing but test the device status while the I/O transaction is in progress. In many cases, the CPU could do useful work in parallel with the I/O transaction, such as:

- computation, as in determining the next output to send to the device or processing the last input received, and
- control of other I/O devices.

To allow parallelism, we need to introduce new mechanisms into the CPU.

The ***interrupt*** mechanism allows devices to signal the CPU and to force execution of a particular piece of code. When an interrupt occurs, the program counter'value is changed to point to an ***interrupt handler***routine (also commonly known as a ***device driver***) that takes care of the device: writing the next data, reading data that have just become ready, and so on. The interrupt mechanism of course saves the value of the PC at the interruption so that the CPU can return to the program that was interrupted. Interrupts therefore allow the flow of control in the CPU to change easily between different ***contexts*,** such as a foreground computation and multiple I/O devices.

The interface between the CPU and I/O device includes the following signals for interrupting:

- the I/O device asserts the **interrupt request** signal when it wants service from the CPU; and
- the CPU asserts the **interrupt acknowledge** signal when it is ready to handle the I/O device's request.

The I/O device's logic decides when to interrupt; for example, it may generate an interrupt when its status register goes into the ready state. The CPU may not be able to immediately service an interrupt request because it may be doing something else that must be finished first—for example, a program that talks to both a high-speed disk drive and a low-speed keyboard should be designed to finish a disk transaction before handling a keyboard interrupt. Only when the CPU decides to acknowledge the interrupt does the CPU change the program counter to point to the device's handler. The interrupt handler operates much like a subroutine, except that it is not called by the executing program. The program that runs when no interrupt is being handled is often called the ***foreground program***; when the interrupt handler finishes, it returns to the foreground program, wherever processing was interrupted.

Before considering the details of how interrupts are implemented, let's look at the interrupt style of processing and compare it to busy-wait I/O. Example 3.4 uses interrupts as a basic replacement for busy-wait I/O; Example 3.5 takes a more sophisticated approach that allows more processing to happen concurrently.

5.Write the sample program to the interrupts routine of RTOS in clear manner?

```
void input_handler()
{ /* get a character and put in global */
achar = peek(IN_DATA);
 /* get character */
 gotchar = TRUE; /*
signal to main program */
poke(IN_STATUS,0);
/* reset status to initiate next transfer */
   }
void output_handler()
{ /* react to character being sent */
```

```
/* don't have to do anything */
}
```

The main program is reminiscent of the busy-wait program. It looks at gotchar to check when a new character has been read and then immediately sends it out to the output device.

```
main() {
while (TRUE) { /* read then write forever */
if (gotchar) { /* write a character */ poke(OUT_DATA,achar); /* put character
in device */
poke(OUT_STATUS,1); /* set status to
initiate write */
gotchar = FALSE; /* reset flag */
}}}
```

The use of interrupts has made the main program somewhat simpler. But this program design still does not let the foreground program do useful work. Example uses a more sophisticated program design to let the foreground program work completely independently of input and output.

6.Write the short notes to the multiple tasks and multiple process in clear manner?

Multiple Tasks and Multiple Processes

processes and operating systems

■The process abstraction.

■Switching contexts between programs.

■Real-time operating systems(RTOSs).

■ Inter process communication.

■Task-level performance analysis and power consumption.

■A telephone answering machine design.

Real-time operating systems (RTOSs), which are Oss that provide facilities for satisfying real-time requirements. ARTOS allocates resources using algorithms that take real time in to account. General-purpose OSs, incontrast, generally allocate resources using other criteria like fairness. Trying to allocate the CPU equally to all processes with out regard to time can easily cause processes to miss their deadlines. In the next section, we will introduce the concepts of task and process. Introduces some basic concepts in inter process communication.

multiple tasks and multiple processes

Most embedded systems require functionality and timing that is too complex to body in a single program. the system in to multiple tasks in order to manage when things happen. To understand why these parathion of an application in to tasks may be reflected in the program structure, consider how we would build a stand-alone compression unit based on the compression algorithm The input to the box is an uncompressed stream of bytes.

The box emits a compressed string of bits on the output serial line, based on a predefined compression table. Such a box may be used, for example, to compress data being sent to a modem. The program's need to receive and send data at different rate for example, the program may emit 2 bits for the first byte and then 7 bits for the second byte will obviously find itself reflected in the structure of the code.

It is easy to create irregular, ungainly code to solve this problem; a more elegant solution is to create a queue of output bits, with those bits being removed from the queue and sent to the serial port in 8-bit sets. But beyond the need to create a clean data structure that simplifies the control structure of the code, that we process the inputs and outputs at the proper rates.

The text compression box provides a simple example of rate control problems. A control panel on a machine provides an example of a different type of rate con- troll problem, the asynchronous input. The control panel of the compression box may, for example, include a compression mode button that disables or enables compression, so that the input text is passed through unchanged when compression is disabled. We certainly do not know when the user will push the compression modebutton the button may be depressed a synchronously relative to the arrival of characters for compression.

That the button will be depressed at a much lower rate than characters will be received, since it is not physically possible for a person to repeatedly depress a button at even slow serial line rates. Keeping up with the input and output data while checking on the button can introduce some very complex control code in to the program. Sampling the button's state too slowly can cause the machine to miss a button depression entirely, but sampling it too frequently and duplicating a data value can cause the machine to in correctly compress data.

One solution is to introduce a counter in to the main compression loop, so that a subroutine to check the input button is called once every times the compression loop is executed. But this solution does not work when either the compression loop or the button-handling routine has highly variable execution times—if the execution time of either varies significantly, it will cause the other to execute later than expected, possibly causing data to be lost. We need to be able to keep track of these two different tasks separately, applying different timing requirements to each.

This is the sort of control that processes allow. The above two examples illustrate how requirements on timing and execution rate can create major problems in programming. When code is written to satisfy several different timing requirements at once, the control structures

necessary to get any sort of solution become very complex very quickly. Worse, such complex control is usually quite difficult to verify for either functional or timing properties.

Multi rate Systems

Implementing code that satisfies timing requirements is even more complex when multiple rates of computation must be handled. Multi rate embedded computing systems are very common, including auto mobile engines, printers, and cell phones.In all these systems, certain operations must be executed periodically, and each oper-ation is executed at its own rate. Application Example6.1 describes why auto mobile engines require multi rate control.

Timing Requirements on Processes

A scheduling policy must define the timing requirements that it uses to determine whether a schedule is valid. Before studying scheduling proper, we outline the types of process timing requirements that are useful in embedded system design

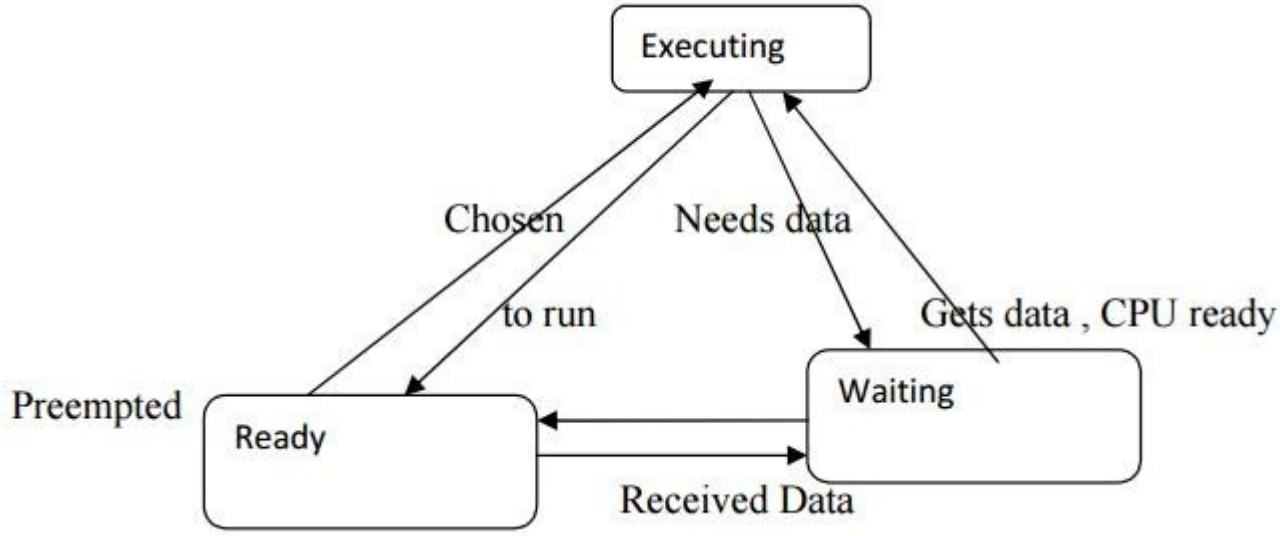
The system can be designed to take a variety of actions when a deadline is missed. Safety-critical systems may try to take compensatory measures such as approximating data or switching in to a special safety mode. Systems for which safety is not as important may take simple measures to avoid propagating bad data, such as inserting silence in a phone line, or may completely ignore the failure.Even if the modules are functionally correct, their timing improper behavior can introduce major execution errors.

Utilization is the ratio of the CPU time that is being used for useful computations to the total available CPUtime. This ratio ranges between 0 and 1, with 1 meaning that all of the available CPU time is being used for system purposes. The utilization is often expressed as a percentage.

7.Write the short notes to the scheduling process in clear manner?

Process State and Scheduling

The work of choosing the order of running processes is known as scheduling. The OS considers a process to be in one of three basic scheduling states Schedulability means whether there exists a schedule of execution for the processes in a system that satisfies all their timing requirements. In general, we must construct a schedule to show schedulability, but in some cases we can eliminate some sets of processes as unschedulable using some very simple tests. Utilization is one of the key metrics in evaluating a scheduling policy. Our most basic requirement is that CPU utilization be no more than 100% since we can’t use the CPU more than100% of the time.



When periodic processes, the length of time that must be considered is the hyper period, which is the least-common multiple of the periods of all the processes. If we evaluate the hyper period, we are sure to have considered all possible combinations of the periodic processes. The next example evaluates the utilization of a simple set of processes.

That some types of timing requirements for a set of processes imply that we can not utilize 100% of the CPU's execution time on useful work, even ignoring context switching overhead. However, some scheduling policies can deliver higher CPU utilizations than others, even for the same timing requirements. The best policy depends on the required timing characteristics of the processes being scheduled.

scheduling policy is known as cyclostatic scheduling or some- times as Time Division Multiple Access scheduling. Processes always run in the same time slot.

Two factors affect utilization: the number of times lots used and the fraction of each time slot that is used for useful work. Depending on the deadlines for some of the processes, we may need to leave some times lots empty. And since the time slots are of equal size, some short processes may have time left over in their time slot.

We can use utilization as a schedulability measure:

the total CPU time of all the processes does not have any useful work to do, the round-robin scheduler moves on to the next process in order to fill the time slot with useful work. In this example, all three processes execute during the first hyper period, but during the second one, P1 has no useful work and is skipped. The processes are always evaluated in the same order. The last time slot in the hyper period is left empty; if we have occasional, non-periodic tasks with out deadlines, we can execute them in these empty time slots.

takes during system operation to implement it. Moreover, we generally achieve higher theoretical CPU utilization by applying more complex scheduling policies with higher over heads. The final decision on a scheduling policy must take in to account both theoretical utilization and practical scheduling overhead.

Running Periodic Processes

We need to find a programming technique that allows us to run periodic processes, ideally at different rates. For the moment, let's think of a process as a subroutine; we will call the mp1(), p2(), etc. for simplicity. Our goal is to run these subroutines at rates determined by the system designer. Here is a very simple program that runs our process subroutines repeatedly: A timer is a much more reliable way to control execution of the loop. We would probably use the timer to generate periodic interrupts. Let's assume for the moment that the pall() function is called by the timer's interrupt handler. Then this code will execute each process once after a timer interrupt:

```
voidpall() {
p1();
p2();
}
```

We could then use a function to collect all the processes that run at that rate:

```
voidpA() {
/*processesthatrunatrateA*/
p1();
p3();
}
voidpB() {
/*processesthatrunatrateB*/
```

This solution allows us to execute processes at rates that are simple multiples of each other. However, when the rates are n't related by a simple ratio, the counting process becomes more complex and more likely to contain bugs. We have developed some what more reliable code, but this programming style is still limited in capability and prone to bugs. To improve both the capabilities and reliability of our systems, we need to invent the RTOS.

8.Write the short notes to the preemptive scheduling process in clear manner?

preemptive real-time operating systems

ARTOS executes processes based upon timing constraints provided by the system designer. The most reliable way to meet timing constraints accurately is to build a preemptive OS and to use priorities to control what process runs at any given time. We will use these two concepts to build up a basic RTOS.

Preemption

Preemption is an alternative to the C function call as away to control execution. To be able to take full advantage of the timer, we must change our notion of a process as something more than a function call. We must, infact, break the assumptions of our high-level programming language. We will create new routines that allow us to jump from one subroutine to another at any point in the program. That, together with the timer, will allow us to move between functions whenever necessary based upon the system's timing constraints.

the CPU a cross two processes. The kernel is the part of the OS that determines what process is running. The kernel is activated periodically by the timer. The length of the timer period is known as the time quantum because it is the smallest increment in which we can control CPU activity. The kernel determines what process will run next and causes that process to run. On the next timer interrupt, the kernel may pick the same processor another process to run.

switch between processes before the process is done? We cannot rely on C-level mechanisms to do so. We can, however, use assembly language to switch between processes. The timer interrupt causes control to change from the currently executing process to the kernel; assembly language can be used to save and restore registers. We can similarly use assembly language to restore registers not from the process that was interrupted by the timer but to use registers from any process we want. These to F registers that define a process are known as its con- text and switching from one process's register set to another is known as context switching. The data structure that holds the state of the process is known as the process control block.

Priorities

each task a numerical priority, then the kernel cans imply look at the processes and their priorities, see which ones actually want to execute (some may be waiting for data or for some event), and select the highest priority process that is ready to run. This mechanism is both flexible and fast. The priority is a non-negative integer value. The exact value of the priority is not as important as the relative priority of different processes.

■A process continues execution until it completes or it is pre empted by a higher-priority process. Let's define a simple system with three processes as seen below. Process Priority

Execution time

P1 1 10

P2 2 30

P3 3 20

P2 is ready to run when the system is started, P1 is released at time 15, and P3 is released at time 18. Once we know the process properties and the environment, we can use the pri-orities to determine which process is running throughout the complete execution of the system. To understand the basics of a context switch, let's assume that these to f tasks is in steady state: Everything has been initialized, the OS is running, and we are ready for a timer interrupt. This diagram shows ,the hardware timer, and all

The functions in the kernel that are involved in the context switch:

```
/*Acknowledge the interrupt at AIC level... */
AT91C_BASE_AIC->AIC_EOICR=portCLEAR_AIC_INTERRUPT;
/*Restore the context of the new task. */
portRESTORE_CONTEXT();
}
```

Pre emptive Tick () has been declared as an aked function; this means that it does not use the normal procedure entry and exit code that is generated by the compiler. Because the function naked, the registers for the process that was interrupted are still available; v Pre emptive Tick () doesn't have to go to the procedure call stack to get their values. This is particularly handy since the procedure mechanism would save only part of the process state, making the state-saving code a little more complex.

The first thing that this routine must do is save the context of the task that was interrupted. To do this, it uses the routine port SAVE_CONTEXT(), which saves all the context of the stack. It the n performs some house keeping, such as incre- menting the tick count. The tick count is the internal timer that is used to determine deadlines. After the tick is incremented, some tasks may have become ready as they passed their deadlines.

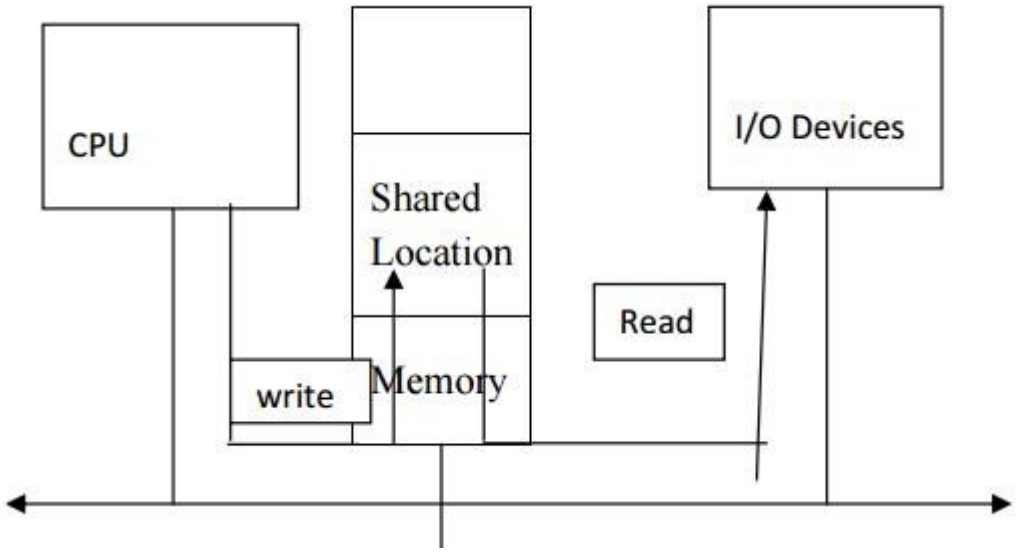
Next, the OS determines which task to run next using the routine vTaskSwitchContext(). After some more house keeping, it uses port RESTORE_CONTEXT() to restore the context of the task that was selected by vTaskSwitchContext(). The action of portRESTORE_CONTEXT() can uses control to transfer to that task with out using the standard C return mechanism.

The code for portSAVE_CONTEXT(), in the file port macro.h, is defined as a macro and not as C function. It is structured in this way so that it does n't disturb the register values that need to be saved. Because it is a macro, it has to be written in a hard-to-read way—all code must be on the same line or end-of-line continuations (back slashes) must be used.

9.Write the short notes to the interprocess communication in clear manner?

interprocess communication mechanisms

Shared Memory Communication:



a process can send a communication in one of two ways: blocking or non blocking. After sending a blocking communication, the process goes in to the waiting state until it receives are sponse. Non blocking communication allows the process to continue execution after sending the communication. Both types of communication are useful.

There are two major styles of inter process communication: shared memory and message passing. The two are logically equivalent—given one, you can build an interface that implements the other. However, some programs may be easier to write using one rather than the other.

The input data arrive at a constant rate and are easy to manage. But because the output data are consumed at a variable rate, these data require an elastic buffer. The CPU and output UART share a memory area the CPU writes compressed characters in to the buffer and the UART removes them as necessary to fill the serial line.

Because the number of bits in the buffer changes constantly, the compression and transmission processes need additional size information. In this case, coordination is simplethe CPU writes at one end of the buffer and the UART reads at the other end. The only challenge is to make sure that the UART does not over run the buffer instruction returns true and the location is in fact set. The bus supports this as an atomic operation that cannot be interrupted.

A test-and-set can be used to implement a semaphore, which is a language-level synchronization construct. For the moment, let’s assume that the system provides one

semaphore that is used to guard access to a block of protected memory. Any process that wants to access the memory must use the semaphore to ensure that no other process is actively using it. As shown below, the semaphore names by tradition are P() to gain access to the protected memory and V() to releaseit.

/* some non protected operations here */

P(); /*wait for semaphore */

/* do protected work here */

V (); /*releases ema phore */

The P () operation uses a test-and-set to repeatedly test a location that holds a lock on the memory block. The P () operation does not exist until the lock is available; once it is available, the test-and-set automatically sets the lock. Once past the P () operation, the process can work on the protected memory block. The V () operation resets the lock, allowing other processes access to the region by using the P() function.

10.Write the short notes to the message passing in clear manner?

Message Passing

Message passing communication complements the shared memory model. The message is not stored on the communications link, but rather at the senders/ receivers at the end points. In contrast, shared memory communication n can be seen as a memory block used as a communication device, in which all the data a restored in the communication link/memory.

Applications in which units operate relatively autonomously are natural candidates for message passing communication. For example, a home control system has one micro controller per house hold device lamp, thermostat, faucet, appliance, and so on. The devices must communicate relatively infrequently; furthermore, their physical separation is large enough that we would not naturally

evaluating operating system performance

The scheduling policy does not tell us all that we would like to know about the performance of a real system running processes. Our analysis of scheduling policies makes some simplifying assumptions:

■We have assumed that context switches require zero time.

processes don't interact, but the cache causes the execution of one program to influence the execution time of other programs. The techniques for bounding the cache-based performance of a single program do not work when multiple programs are in the same cache. Many real-time systems have been designed based on the assumption that there is no cache present, even though one actually exists. This grossly conservative assumption is made because the system architects lack tools that permit them to analyze the effect of caching.

Since they do not know where caching will cause problems, they are forced to retreat to the simplifying assumption that there is no cache. The result is extremely over designed hardware, which has much more computational power than is necessary.

- However, just as experience tells us that a well-designed cache provides significant performance benefits for a single program, a properly sized cache can allow a microprocessor to run a set of processes much more quickly. By analyzing the effects of the cache, we can make much better use of the available hardware.
- some processes can be given reservations in the cache, such that only a particular process can inhabit a reserved section of the cache; other processes are left to share the cache. We generally want to use cache partitions only for performance-critical processes in cache reservations are wasteful of limited cache space
- Performance is estimated by constructing a schedule, taking in to account not just execution time of the processes but also the state of the cache.
- Each process in the shared section of the cache is modeled by a binary variable: 1 if present in the cache and 0 if not. Each process is also characterized by three total execution times: assuming no caching, with typical caching, and with all code always resident in the cache. The always-resident time is unrealistically optimistic, but it can be used to find a lower bound on the required schedule time. During construction of the schedule, we can look at the current cache state to see whether the no-cache or typical-caching execution time should be used at this point in the schedule.
- update the cache state if the cache is needed for another process. Although this model is simple, it provides much more realistic performance estimates than assuming the cache either is none x is tent or is perfect
- Because power-down and power-up are not free, modes should be changed carefully. Determining when to switch into and out of a power-up mode requires an analysis of the overall system activity.

■Avoiding a power-down mode can cost un necessary power.

■Powering down too soon can cause severe performance penalties.

Re-entering run mode typically costs a considerable amount of time.

A straight forward method is to power up the system when a request is received. This works as long as the delay in handling the request is acceptable. A more sophisticated technique is predictive shutdown.

The goal is to predict when the next request will be made and to start the system just before that time, saving their quest or the start-up time. In general, predictive shutdown techniques are probabilistic they make guesses about activity patterns based on a proba-

bilistic model of expected behavior. Because they rely on statistics, they may not always correctly guess the time of the next activity.

11.Write the short notes about the coding of VxWorks in clear manner?

Coding using VxWorks Adapted toto OSEK-OS Features

CHALLENGES IN EMBEDDED COMPUTING SYSTEM DESIGN

The following external constraints are one important source of difficulty in embedded system Design.

- **How much hardware do we need?**

We have a great deal of control over the amount of computing power we apply to our problem. We cannot only select the type of microprocessor used, but also select the amount of memory, the peripheral devices, and more. If too little hardware and the system fails to meet its deadlines, too much hardware and it becomes too expensive.

- **How do we meet deadlines?**

It is entirely possible that increasing the CPU clock rate may not make enough difference to execution time, since the program's speed may be limited by the memory system.

- **How do we minimize power consumption?**

In battery-powered applications, power consumption is extremely important. Even in non-battery applications, excessive power consumption can increase heat dissipation. One way to make a digital system consume less power is to make it run more slowly, but naively slowing down the system can obviously lead to missed deadlines.

- **How do we design for upgradability?**

The hardware platform may be used over several product generations or for several different versions of a product in the same generation, with few or no changes.

- **Does it really work?**

Reliability is always important when selling products—customers rightly expect that products they buy will work.

The sources that make the design so difficult are:

■ ***Complex testing****:* Exercising an embedded system is generally more difficult than typing in some data. The timing of data is often important, meaning that we cannot separate the testing of an embedded computer from the machine in which it is embedded.

Limited observability and controllability*:* Embedded computing systems usually do not come with keyboards and screens. This makes it more difficult to see what is going on and to affect the system's operation. We may be forced to watch the values of electrical signals on the microprocessor bus, for example, to know what is going on inside the system. Moreover, in real-time applications we may not be able to easily stop the system to see what is going on inside.

■ ***Restricted development environments****:* The development environments for embedded systems (the tools used to develop software and hardware) are often much more limited than those available for PCs and workstations.

be used and how the

12.Explain the SmartOS RTOS used as alternative to MUCOS in detailed manner?

Smart Card OS

- SmartOS─ assumed hypothetical OS in this example, as RTOS in the card.
- Use for understanding purposes identical to MUCOS but actual SmartOS has to be different from MUCOS.
- Its file structure is different, though it has MUCOS like IPCs and ECBs.
- function unsigned char [ ] SmartOSEncrypt (unsigned char *applStr, EnType type) encrypts as per encryption method, EnType = "RSA" or "DES" algorithm chosen and returns the encrypted string
- function unsigned char [ ] SmartOSDecrypt (unsigned char *Str, DeType type) encrypts as per deciphering method, DeType = "RSA" or "DES" algorithm chosen and returns the deciphered string.

SmartOSEncrypt and SmartOSDecrypt execute after verifying the access conditions from the data files that store the keys, PIN (Personal Identification Number) and password.

13.Explain the memory system mechanishm in detailed manner?

Modern microprocessors do more than just read and write a monolithic memory. Architectural features improve both the speed and capacity of memory systems. Microprocessor clock rates are increasing at a faster rate than memory speeds, such that memories are falling further and further behind microprocessors every day.

As a result, computer architects resort to ***caches*** to increase the average performance of the memory system. Although memory capacity is increasing steadily, program sizes are increasing as well, and designers may not be willing to pay for all the memory demanded by an application.

Modern microprocessor units (MMUs) perform address translations that provide a larger virtual memory space in a small physical memory. In this section, we review both caches and MMUs.

Caches

Caches are widely used to speed up memory system performance. Many microprocessors architectures include caches as part of their definition. The cache speeds up average memory access time when properly used.

It increases the variability of memory access times—accesses in the cache will be fast, while access to locations not cached will be slow. This variability in performance makes it especially important to understand how caches work so that we can better understand how to predict cache performance and factor variability's into system design.

A cache is a small, fast memory that holds copies of some of the contents of main memory. Because the cache is fast, it provides higher-speed access for the CPU; but since it is small, not all requests can be satisfied by the cache, forcing the system to wait for the slower main memory. Caching makes sense when the CPU is using only a relatively small set of memory locations at any one time; the set of active locations is often called the ***working set***.

If the requested location is in the cache, the cache controller forwards the location's contents to the CPU and aborts the main memory request; this condition is known as a ***cache hit***. If the location is not in the cache, the controller waits for the value from main memory and forwards it to the CPU; this situation is known as a ***cache miss***.

- A ***compulsory miss*** (also known as a **cold miss**) occurs the first time a location is used, A ***capacity miss*** is caused by a too-large working set, and
- A ***conflict miss*** happens when two locations map to the same location in the cache.

Even before we consider ways to implement caches, we can write some basic formulas for memory system performance. Let *h* be the ***hit rate***, the probability that a given memory location is in the cache. It follows that 1_*h* is the ***miss rate***, or the probability that the location is not in the cache. Then we can compute the average memory access time as

Where *t*cache is the access time of the cache and *t*main is the main memory access time. The memory access times are basic parameters available from the memory manufacturer. The hit rate depends on the program being executed and the cache organization, and is typically measured using simulators. The best-case memory access time (ignoring cache controller overhead) is *t*cache, while the worst-case access time is*t*main. Given that *t*main is typically 50– 60 ns for DRAM, while *t*cache is at most a few nanoseconds, the spread between worst-case and best-case memory delays is substantial.

The second-level cache is much larger but is also slower. If *h*1 is the first-level hit rate and *h*2 is the rate at which access hit the second-level cache but not the first-level cache, then the average access time for a two-level cache system is

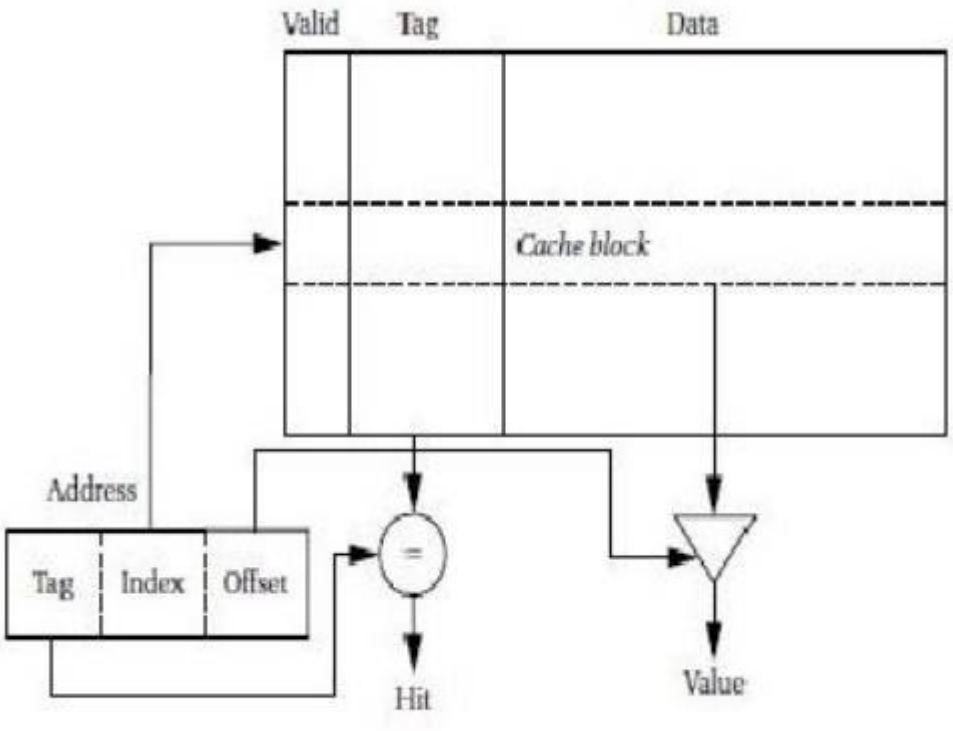
As the program's working set changes, we expect locations to be removed from the cache to make way for new locations. When set-associative caches are used; we have to think about what happens when we throw out a value from the cache to make room for a new value. We do not have this problem in direct-mapped caches because every location maps onto a unique block, but in a set-associative cache we must decide which set will have its block thrown out to make way for the new block. One possible replacement policy is least

recently used (LRU), that is, throw out the block that has been used farthest in the past. We can add relatively small amounts of hardware to the cache to keep track of the time since the last access for each block. Another policy is random replacement, which requires even less hardware to implement.

The simplest way to implement a cache is a ***direct-mapped cache***, as shown in Figure. The cache consists of cache ***blocks***, each of which includes a tag to show which memory location is represented by this block, a data field holding the contents of that memory, and a valid tag to show whether the contents of this cache block are valid. An address is divided into three sections. The index is used to select which cache block to check. The tag is compared against the tag value in the block selected by the index. If the address tag matches the tag value in the block, that block includes the desired memory location. If the length of the data field is longer than the minimum addressable unit, then the lowest bits of the address are used as an offset to select the required value from the data field. Given the structure of the cache, there is only one block that must be checked to see whether a location is in the cache —the index uniquely determines that block. If the access is a hit, the data value is read from the cache.

Writes are slightly more complicated than reads because we have to update main memory as well as the cache. There are several methods by which we can do this. The simplest scheme is known as ***write-through***—every write changes both the cache and the corresponding main memory location (usually through a write buffer).

This scheme ensures that the cache and main memory are consistent, but may generate some additional main memory traffic. We can reduce the number of times we write to main memory by using a ***write-back*** policy: If we write only when we remove a location from the cache, we eliminate the writes when a location is written several times before it is removed from the cache.



The direct-mapped cache is both fast and relatively low cost, but it does have limits in its caching power due to its simple scheme for mapping the cache onto main memory. Consider a direct-mapped cache with four blocks, in which locations 0, 1, 2,and 3 all map to different blocks. But locations 4, 8,12,…all map to the same block as location 0;locations 1, 5, 9,13,…all map to a single block; and so on.

If two popular locations in a program happen to map onto the same block, we will not gain the full benefits of the cache. As seen in Section 5.6, this can create program performance problems.

The set-associative cache generally provides higher hit rates than the direct mapped cache because conflicts between a small numbers of locations can be resolved within the cache.

The set-associative cache is somewhat slower, so the CPU designer has to be careful that it doesn't slow down the CPU's cycle time too much. A more important problem with set-associative caches for embedded program design is predictability.

UNIT V

1.Explain the Smart Card System applications in detail manner?

Smart Card System Requirements

Purpose Purpose Enabling authentication and verification of card and card holder by a host Enabling GUI at host machine to interact with the card holder/user for the required transactions, for example, financial transactions with a bank or credit card transactions.

Inputs Received header and messages at IO port Port_IO from host through the antenna

Internal Signals, Events and Internal Signals, Events and Notifications Notifications On power up, radiation-powered chargepump supply of the card activated and a signal to start the system boot program at resetTask Card start requestHeader message to task_ReadPort from resetTask Host authentication request requestStart message to task_ReadPort from resetTask to enable requests for Port_IO

UserPW verification message (notification) through Port_IO from host Card application close request requestApplClose message to Port_IO

Outputs Outputs Transmitted headers and messages at Port_IO through antenna

No control panel is at the card. The control panel and GUIs activate at the host machine (for example, at ATM or credit card reader)

The card inserts at a host machine. The radiations from the host activate a charge pump at the card. The charge pump powers the SoC circuit consisting of card processor, memory, timer, interrupt handler and IO port, Port_IO. On power up, system reset signals resetTask to start. The resetTask sends the messages requestHeader and requestStart for waiting task task_ReadPort.task_ReadPort sends requests for host identification and reads through the Port_IO the host-identification message and request for card identification.
task_PW sends through Port_IO the requested card identification after system receives the host identity through Port_IO. task_Appl then runs required API. The requestApplClose message closes the application.

Test and validation conditions Test and validation conditions Tested on different host machine versions for fail proof card-host communication.

Classes

Task_CardCommunication is an abstract class from which extended to class (es) derive to read port and authenticate. The tasks (objects) are the instances of the classes Task_Appl, Task_Reset, Task_ReadPort and Task_PW. ISR1_Port_IO, ISR2_Port_IO and ISR3_Port_IO are interfaces to the tasks

Other Classes Other Classes

Classes for the network, sockets, connections, datagrams, character-input output and streams, security management, digital-certification, symmetric and asymmetric keys-based cryptography and digital signatures.

2.Explain the_Smart Card System functional components_in detail manner?

Smart Card Hardware Smart Card Hardware

• A plastic card in ISO standard dimensions, 85.60 mm x 53.98 x 0.80 mm. It is an embedded SoC (System-OnChip). [ISO standards - ISO7816 (1 to 4) for host-machine contact based card and ISO14443 (Part A or B) for the contactless cards.]

Microcontroller MC68HC11D0 or PIC16C84 or a smart card processor Philips Smart XA or an ASIP Processor. Needs 8 kB+ internal RAM and 32 kB EPROM and 2/3 wire protected memory. • CPU special features, for example, a security lock

Smart Card Hardware … Smart Card Hardware …

• CPU locks certain section of memory - protect 1 kB or more data from modification and access by any external source or instruction outside that memory

• Other way of protecting - CPU access through the physical addresses, which are different from logical address used in the program.

Standard ROM 8 kB for usual or 64 kB when using advanced cryptographic features

• Full or part of ROM bus activates take place after a security check only.

ROM Contains: ROM Contains:

iFabrication key and Personalisation key (after insertion of this key, RTOS and application use only the logical addresses)

ii. RTOS codes

iii. Application codes iv. Utilisation lock

EEPROM or Flash scalable – only needed part unlocks when storing P.I.N., unlocking P.I.N., access condition, card-user data, post activation application run generated non-volatile data, invalidation lock to invalidate card after the expiry date or server instruction

Smart Card Hardware

• RAM – run time temporary variables

• Chip-supply system using charge pump

• I/O system

Smart Card Software Smart Card Software

• Needs cryptographic software, needs special features in its operating system over and above the MS DOS or UNIX system features.

Protected environment -OS stored in the protected part of ROM.

• A restricted run-time environment.

• OS, every method, class and run time library should be scalable

Optimum Code-size

• Limited use of data types; multidimensional arrays, long 64-bit integer and floating points and very limited use of the error handlers, exceptions, signals, serialisation, debugging and profiling

Three-layered file system for the data

• master file to store all file headers (file status, access conditions and the file lock)

A header means file status, access conditions and the file lock.

• Dedicated file ─ second file to hold a file grouping and headers of the immediate successor

• Elementary file ─ third file to hold the file header and its file data.

3.Write short notes about the washing machine in detail manner?

General durability issues in washing machines Washing machines are constructed with the same basic components; the agitator, motor, drive (belt or shaft), water feed tube, wash tub (with inner and outer casing) and drain tube.

Product variation is demonstrated by the positioning of these parts, although the fundamental differentiator that contributes to durability and repair in washing machines is the quality of their build and components.

Price-point differentiation for consumers however, tends to focus on energy rating, features such as wash programme variations (including maximum spin speed), drum size, sensor controls and type and number of displays.

A UK study on washing machines1 found that 32% of machines in use have undergone some level of repair. On average the age ofthe washing machines was found to be 5.5 years, and the average age of those that had been regularly serviced was 6.8 years - indicating that servicing and repair results in product life extension. According to the study the expected lifetime of a washing machine is 10 years or more. Bosch has found that electrical failure is currently the leading fault, particularly of the PCB (printed circuit board) caused by fluctuations in mains voltage supply, although surge protection is provided with these machines.

Electrical faults can also occur as a result of water leaks from poor installation of the machine in the household and blockages in the soap drawer or inlet and outlet pipes. However Bosch have found that failures of other electrical components such as motors and pumps are becoming less common.

Research with washing machine manufacturers and the repair industry found that parts that are more prone to wear and that are more likely to need replacing are: door seals and hinges (items becoming caught in the seals, orν deterioration of the rubber); inlet and outlet hoses;ν water heating elements;ν drum bearings (failure due to water leaks);ν motors (particularly from wear on brushes);ν soap drawer (misuse, or detergent solidifying causingν blockages); and motor and drum bearings (due to overloading).

Electrical robustness Motors Siemens use an innovative motor technology to improve durability as well as energy efficiency and speed, and an anti-vibration system to increase stability and reduce damage.

This is achieved by an entirely enclosed brushless motor (as brushes wear out and need replacement), which the company believe is their most durable motor. Leak protection Most of the electronics are located towards the top of the machines to prevent water damage.

The electronic components below the tub are covered, but could still be at risk from water damage in the event of a leak. In both machines the wiring looms and connectors are positioned to minimise electrical failure.

The motor cabling in the Bosch model is protected from potential leaks by a plastic shroud. 8 Component and cable fixings The PCBs inside both machines are secured with

clips that enable quick and easy replacement over numerous access cycles – and these fixings may also dampen and resist vibration more effectively than screws giving greater security over time.

All electrical connectors are secured firmly with snap-fits that resist vibration. The use of plastic connectors rather than soldered joints also allows easier access for parts.

4.Write short notes about the ACVM in detail manner?

ACVM Coin insertion slot Keypad on the top of the machine. LCD display unit on the top of the machine. It displays menus, text entered into the ACVM and pictograms, welcome, thank and other messages. Graphic interactions with the machine. Displays time and date.

Delivery slot so that child can collect the chocolate and coins, if refunded. Internet connection port so that owner can know status of the ACVM sales from remote

ACVM Hardware units ACVM Hardware units

Microcontroller or ASIP (Application Specific Instruction Set Processor) RAM for storing temporary variables and stack ROM for application codes and RTOS codes for scheduling the tasks Flash memory for storing user preferences, contact data, user address, user date of birth, user identification code, answers of FAQsTimer and Interrupt controller A TCP/IP port (Internet broadband connection) to the ACVM for remote control and for getting ACVM status reports by owner. ACVM specific hardware Power supply .

ACVM Software components

Keypad input read Display Read coins Deliver chocolate TCP/IP stack processing TCP/IP stack communication

Alphanumeric keypad and Display

Alphanumeric keypad on the top of the machine. A child interaction with it when buying a chocolate. Owner commands and interaction with the machine. Three line LCD display unit on the top of the machine. Displays menus, entered text, pictograms, and welcome, thank and other messages, and time and date. Child as well as the ACVM owner GUIs with the machine using keypad and display.

Coin insertion and delivery slots,

Internet port Internet port Coin insertion slot so that the child can insert the coins to buy a chocolate Delivery slot to collect the chocolate, and coins if refunded Internet connection port so that owner can interact with ACVM from remote.

5.Explain the_Smart Card_software functional components_in detail manner?

Smart Card Hardware

- A plastic card in ISO standard dimensions, 85.60 mm x 53.98 x 0.80 mm. It is an embedded SoC (System-On-Chip). [ISO standards - ISO7816 (1 to 4) for host-machine contact based card and ISO14443 (Part A or B) for the contact-less cards.]
- Microcontroller MC68HC11D0 or PIC16C84 or a smart card processor Philips Smart XA or an ASIP Processor. Needs 8 kB+ internal RAM and 32 kB EPROM and 2/3 wire protected memory.
- CPU special features, for example, a security lock
- CPU locks certain section of memory - protect 1 kB or more data from modification and access by any external source or instruction outside that memory
- Other way of protecting - CPU access through the physical addresses, which are different from logical address used in the program.
- Standard ROM 8 kB for usual or 64 kB when using advanced cryptographic features
- Full or part of ROM bus activates take place after a security check only.

ROM Contains:

i. Fabrication key and Personalisation key (after insertion of this key, RTOS and application use only the logical addresses)

ii. RTOS codes

iii. Application codes

iv. Utilisation lock

- EEPROM or Flash scalable – only needed part unlocks when storing P.I.N., unlocking P.I.N., access condition, card-user data, post activation application run generated non-volatile data, invalidation lock to invalidate card after the expiry date or server instruction
- RAM – run time temporary variables
- Chip-supply system using charge pump
- I/O system

Software Architecture

Smart Card Software

- Needs cryptographic software, needs special features in its operating system over and above the MS DOS or UNIX system features.
- Protected environment -OS stored in the protected part of ROM.

- A restricted run-time environment.
- OS, every method, class and run time library should be scalable.
- Optimum Code-size
- Limited use of data types; multidimensional arrays, long 64-bit integer and floating points and very limited use of the error handlers, exceptions, signals, serialisation, debugging and profiling
- Three-layered file system for the data
- master file to store all file headers (file status, access conditions and the file lock)
- A header means file status, access conditions and the file lock.

- Dedicated file─ second file to hold a file grouping and headers of the immediate successor
- Elementary file ─ third file to hold the file header and its file data.
- Either a fixed length file management or a variable file length management with each file with a predefined offset.
- Java CardTM, EmbeddedJava or J2ME (Java 2 Micro Edition) JVM has thread scheduler built in.
- Java provides the features to support (i) security using class java.lang.SecurityManager), (ii) cryptographic needs (package java.security*).