DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

EC8791 EMBEDDED AND REAL TIME SYSTEMS L T P C 3 0 0 3

Objectives:

- To Understand the concept of embedded system design and analysis.
- To learn the architecture of ARM processor.
- To learn the Programming of ARM processor
- To Expose the basic concepts of embedded programming.
- To Learn real time operating systems

UNIT I INTRODUCTION TO EMBEDDED SYSTEM DESIGN

Complex systems and microprocessors– Embedded system design process –Design example: Model train controller- Design methodologies- Design flows - Requirement Analysis – Specifications-System analysis and architecture design – Quality Assurance techniques - Designing with computing platforms – consumer electronics architecture – platform-level performance analysis.

UNIT II ARM PROCESSOR AND PERIPHERALS

ARM Architecture Versions – ARM Architecture – Instruction Set – Stacks and Subroutines – Features of the LPC 214X Family – Peripherals – The Timer Unit – Pulse Width Modulation Unit – UART – Block Diagram of ARM9 and ARM Cortex M3 MCU.

UNIT III EMBEDDED PROGRAMMING

Components for embedded programs- Models of programs- Assembly, linking and loading – compilation techniques- Program level performance analysis – Software performance optimization – Program level energy and power analysis and optimization – Analysis and optimization of program size- Program validation and testing.

UNIT IV REAL TIME SYSTEMS

Structure of a Real Time System — Estimating program run times – Task Assignment and Scheduling – Fault Tolerance Techniques – Reliability, Evaluation – Clock Synchronisation..

UNIT V PROCESSES AND OPERATING SYSTEMS

Introduction – Multiple tasks and multiple processes – Multirate systems- Preemptive realtime operating systems- Priority based scheduling- Interprocess communication mechanisms – Evaluating operating system performance- power optimization strategies for processes – Example Real time operating systems-POSIX-Windows CE. - Distributed embedded systems – MPSoCs and shared memory multiprocessors. – Design Example - Audio player, Engine control unit – Video accelerator.

TOTAL : 45 PERIODS

Course Outcomes:

At the end of the course, the student will be able to:

- **Summarize** Architecture and programming of ARM processor.
- **Applying** the concepts of embedded systems and its features.
- **Analyze** various Real Time Operating system is used in Embedded System.
- **Design** the flow &Techniques to develop Software for embedded system networks.
- **Analyze** Real-time applications using embedded System Products.

TEXT BOOK:

1. Marilyn Wolf, "Computers as Components - Principles of Embedded Computing System Design", Third Edition "Morgan Kaufmann Publisher (An imprint from Elsevier), 2012. (UNIT I, II, III, V)
2. Jane W.S.Liu,‖ Real Time Systems‖, Pearson Education, Third Indian Reprint, 2003.(UNIT IV)

REFERENCES:

1. Lyla B.Das, ―Embedded Systems : An Integrated Approach‖ Pearson Education, 2013.
2. Jonathan W.Valvano, "Embedded Microcomputer Systems Real Time Interfacing", Third Edition Cengage Learning, 2012.
3. David. E. Simon, "An Embedded Software Primer", 1st Edition, Fifth Impression, Addison-Wesley Professional, 2007.
4. Raymond J.A. Buhr, Donald L.Bailey, "An Introduction to Real-Time Systems- From Design to Networking with C/C++", Prentice Hall, 1999.
5. C.M. Krishna, Kang G. Shin, "Real-Time Systems", International Editions, Mc Graw Hill 1997
6. K.V.K.K.Prasad, "Embedded Real-Time Systems: Concepts, Design & Programming", Dream Tech Press, 2005.
7. Sriram V Iyer, Pankaj Gupta, "Embedded Real Time Systems Programming", Tata Mc Graw Hill, 2004.

EC8791 -EMBEDDED AND REAL TIME SYSTEMS

1. Wayne Wolf, “Computers as Components – Principles of Embedded Computing System Design”, Third Edition “Morgan Kaufmann Publisher (An imprint from Elsevier), 2012.

UNIT I

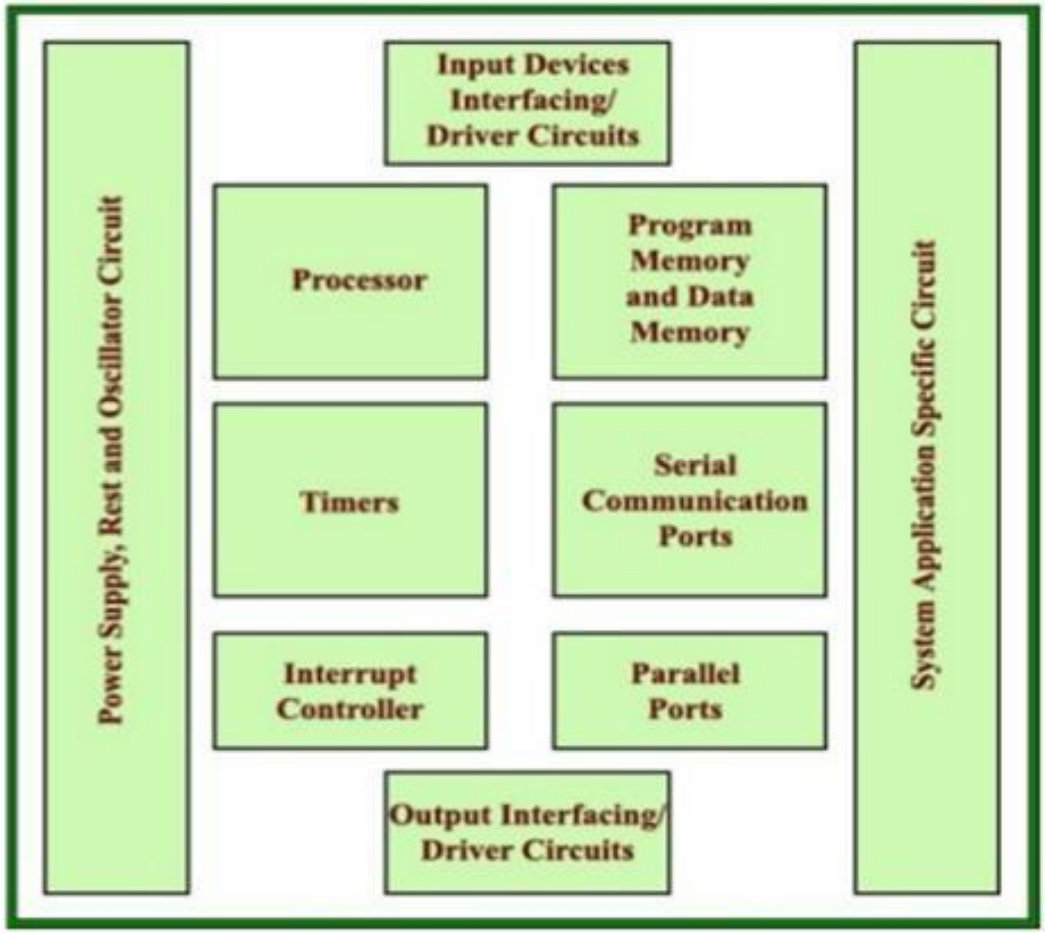
INTRODUCTION TO EMBEDDEDSYSTEM DESIGN

Complex systems and microprocessors– Embedded system design process –Design example: Model train controller- Design methodologies- Design flows - Requirement Analysis – Specifications-System analysis and architecture design – Quality Assurance techniques - Designing with computing platforms – consumer electronics architecture – platform-level performance analysis.

Introduction-Embedded Systems

- An Embedded system is an electronic system that has a software and is embedded in computer hardware.
- It is a system which has collection of components used to execute a task according to a program or commands given to it.
- Examples →Microwave ovens, Washing machine, Telephone answering machine system, Elevator controller system, Printers, Automobiles, Cameras, etc.

EMBEDDED SYSTEM HARDWARE



Components of Embedded system

- Microprocessor
- Memory Unit(RAM,ROM)
- Input unit(Keyboard,mouse,scanner)
- Output unit(pinters,video monitor)
- Networking unit(Ethernet card)
- I/O units(modem)

Real Time Operating System-RTOS

- Real-Time Operating System (**RTOS**) is an operating system (OS) intended to serve real-time applications that process data as it comes in, typically without buffer delays.
- It schedules their working and execution by following a plan to control the latencies and to meet the dead lines.
- Modeling and evaluation of a real-time scheduling system concern is on the analysis of the algorithm capability to meet a process deadline.
- A deadline is defined as the time required for a task to be processed.

Classification of Embedded system

1. Small scale Embedded system→(8/16bit microcontroller)
2. Medium Scale Embedded system→ →(16/32bit microcontroller, more tools like simulator, debugger)
3. Sophisticated Embedded system →(configurable processor and PAL)

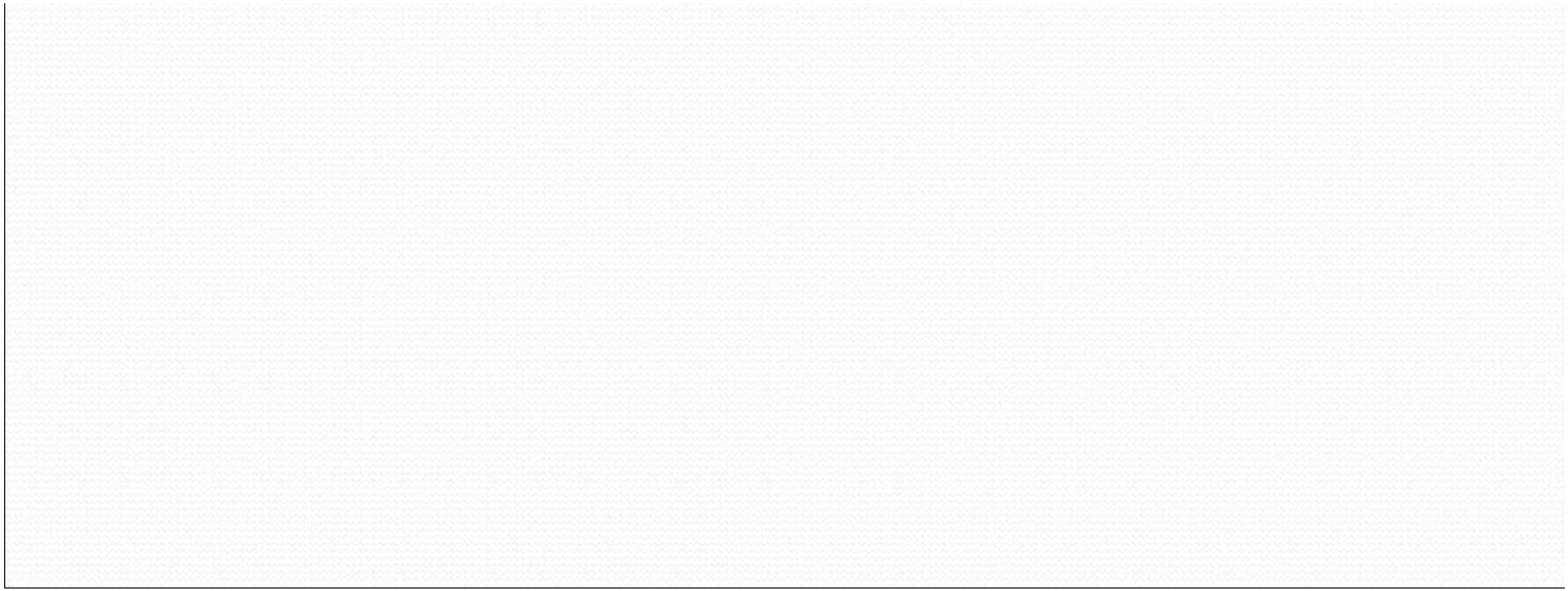
Embedded designer-skills

- Designer has a knowledge in the followings field,
- Microcontrollers, Data comm., motors, sensors, measurements ,C programming, RTOS programming.

1) COMPLEX SYSTEMS AND MICROPROCESSORS

Embedded(+)computer system

- Embedded system is a complex system
- It is any device that includes a programmable computer but is not itself intended to be a general-purpose computer.



History of Embedded computer system

- Computers have been embedded into applications since the earliest days of computing.
- In 1940s and 1950s→Whirlwind, designed a first computer to support real-time operation for controlling an aircraft simulator.
- In 1970s→ The first microprocessor(Intel 4004) was designed for an embedded application (Calculator), provided basic arithmetic functions.
- In 1972s→ The first handheld calculator (HP-35) was to perform transcendental functions , so it used several chips to implement the CPU, rather than a single-chip microprocessor.
- Designer faced critical problems to design a digital circuits to perform operations like trigonometric functions using calculator.
- But ,Automobile designers started making use of the microprocessor for to control the engine by determining when spark plugs fire, controlling the fuel/air mixture

Levels of Microprocessor

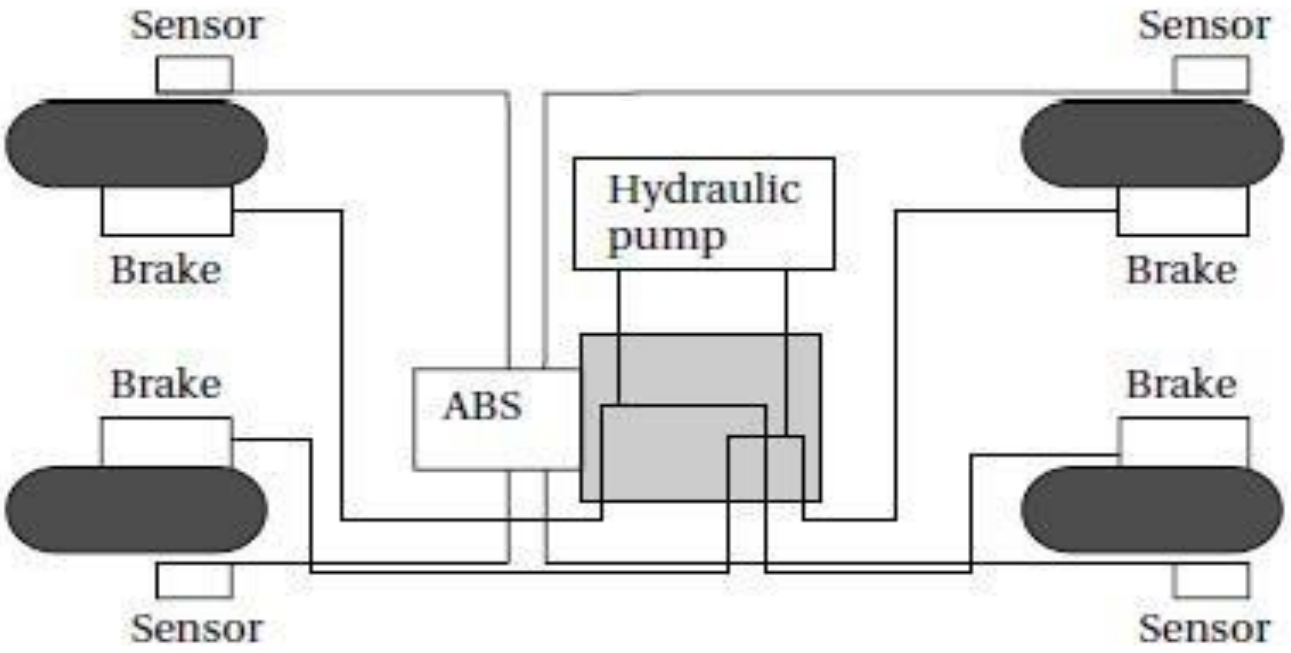
1. 8-bit ***microcontroller*** → for low-cost applications and includes on-board memory and I/O devices.
2. 16-bit microcontroller → used for more sophisticated applications that may require either longer word lengths or off-chip I/O and memory.
3. 32-bit ***RISC*** microprocessor →offers very high performance for computation-intensive applications.

Microprocessor Uses/Applications

- Microwave oven has at least one microprocessor to control oven operation
- Thermostat systems, which change the temperature level at various times during the day
- The modern camera is a prime example of the powerful features that can be added under microprocessor control.
- Digital television makes extensive use of embedded processors.

Embedded Computing Applications

- Ex→BMW 850i Brake and Stability Control System
- The BMW 850i was introduced with a sophisticated system for controlling the wheels of the car.
- Which uses An antilock brake system (ABS) and An automatic stability control (ASC +T) system.



1. An antilock brake system (ABS)

- Reduces skidding by pumping the brakes.
- It is used to temporarily release the brake on a wheel when it rotates too slowly—when a wheel stops turning, the car starts skidding and becomes hard to control.
- It sits between the hydraulic pump, which provides power to the brakes.
- It uses sensors on each wheel to measure the speed of the wheel.
- The wheel speeds are used by the ABS system to determine how to vary the hydraulic fluid pressure to prevent the wheels from skidding.

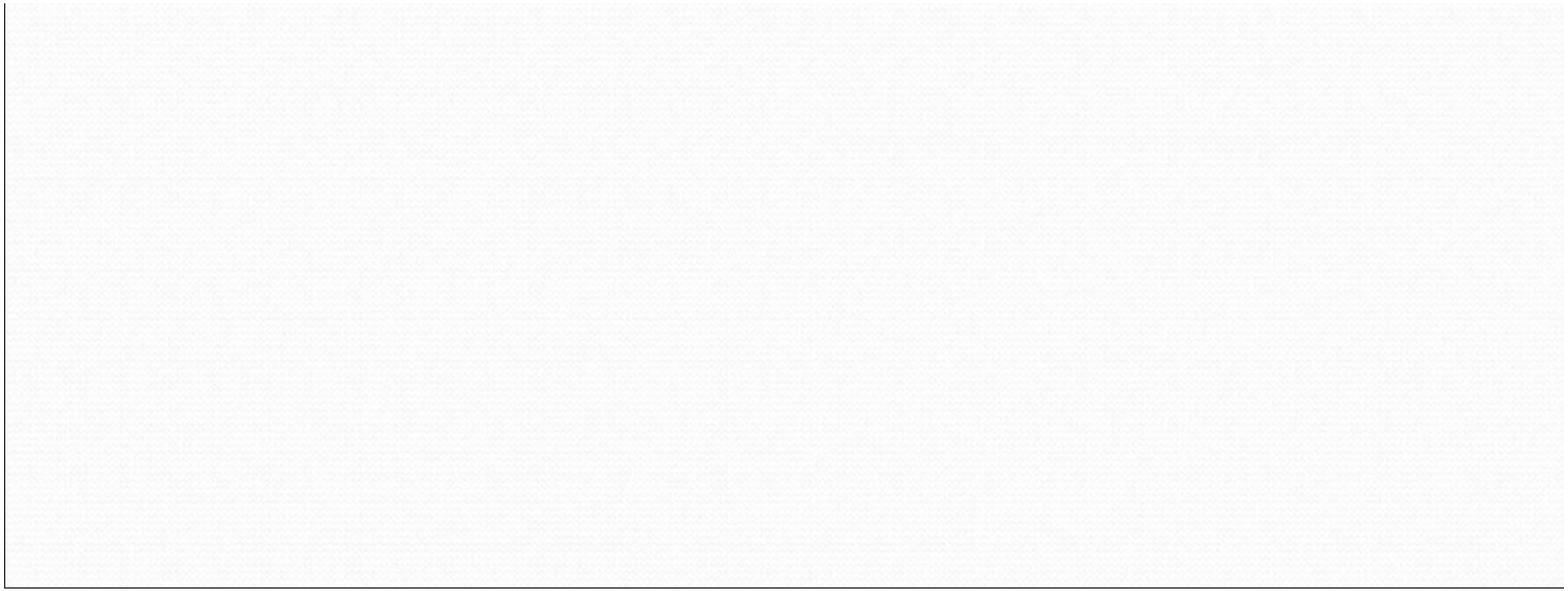
2. An automatic stability control (ASC +T) system

- It is used to control the engine power and the brake to improve the car's stability during maneuvers.
- It controls four different systems: throttle, ignition timing, differential brake, and (on automatic transmission cars) gear shifting.
- It can be turned off by the driver, which can be important when operating with tire snow chains.
- It has control unit has two microprocessors , one of which concentrates on logic-relevant components and the other on performance-specific components.
- The ABS and ASC+ T must clearly communicate because the ASC+ T interacts with the brake system.

Characteristics of Embedded Computing Applications

1. Complex algorithms-The microprocessor that controls an automobile engine must perform complicated filtering functions to optimize the performance of the car while minimizing pollution and fuel utilization.
2. User interface-The moving maps in Global Positioning System (GPS) navigation are good examples of user interfaces.
3. Real time-Embedded computing systems have to perform in real time—if the data is not ready by a certain deadline,the system breaks. In some cases, failure to meet a deadline or missing a deadline does not create safety problems but does create unhappy customers
4. Multirate-Multimedia applications are examples of ***multirate*** behavior. The audio and video portions of a multimedia stream run at very different rates, but they must remain closely synchronized. Failure to meet a deadline on either the audio or video portions spoils the perception of the entire presentation.

5. Manufacturing cost- It is depends on the type of microprocessor used, the amount of memory required, and the types of I/O devices.
6. Power and energy-Power consumption directly affects the cost of the hardware, since a larger power supply may be necessary.
7. Energy consumption →affects battery life, which is important in many applications, as well as heat consumption, which can be important even in desktop applications.

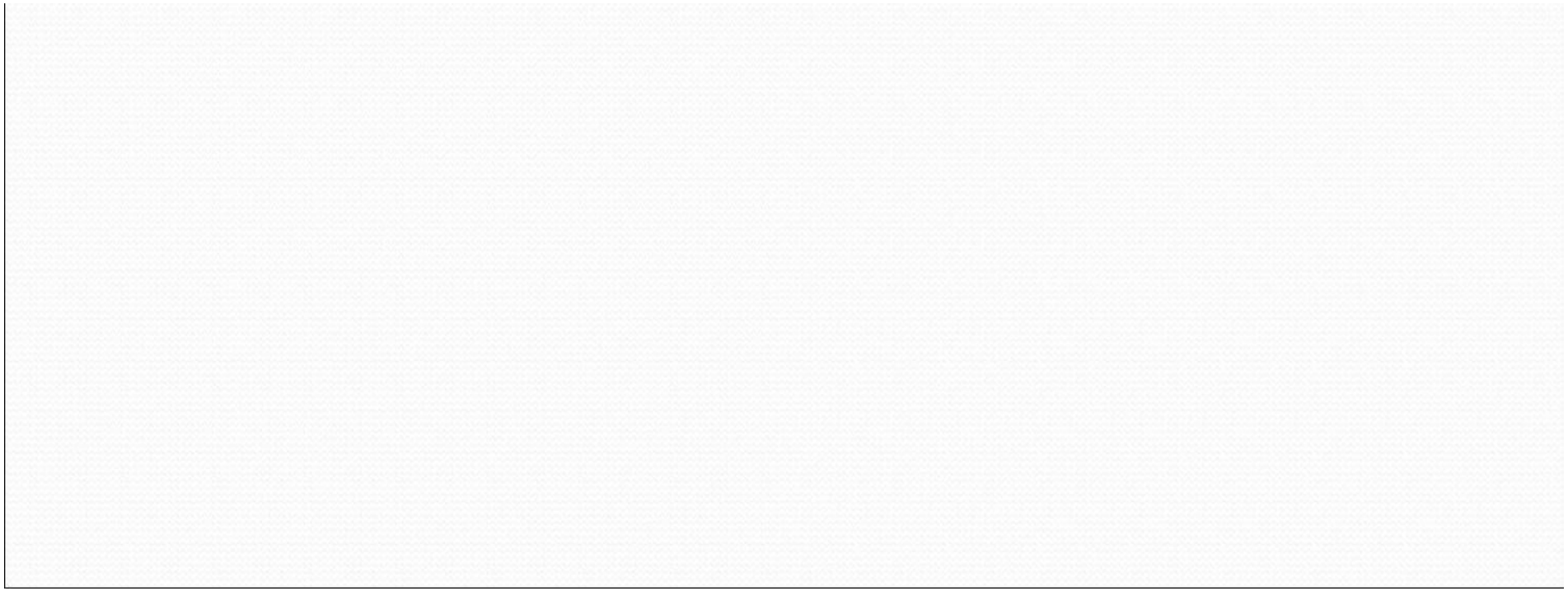


Why Use Microprocessors?

- Microprocessors are a very efficient way to implement digital systems.
- It make it easier to design families of products with various feature at different price points
- It can be extended to provide new features to keep up with rapidly changing markets.
- It executes program very efficiently
- It make their CPU run very fast
- Implementing several function on a single processor

Why not use PCs for all embedded computing?

- Real time performance is very less in PC because of different architecture.
- It increases the complexity and price of components due to broad mix of computing requirements.



Challenges in Embedded Computing System Design

1. How much hardware do we need?

- To meet performance deadlines and manufacturing cost constraints, the choice of Hardware is important.
- Too much hardware and it becomes too expensive.

2. How do we meet deadlines?

- To speed up the hardware so that the program runs faster. But the system more expensive.
- It is also 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.

3. How do we minimize power consumption?

- In battery-powered applications, power consumption is extremely important.
- In non-battery applications, excessive power consumption can increase heat dissipation.
- Careful design is required to slow down the noncritical parts of the machine for power consumption while still meeting necessary performance goals.

4) How do we design for upgradability?

- The hardware platform may be used over several product generations,or for several different versions ,able to add features by changing software.

  Complex testing: Run a real machine in order to generate the proper data.

- Testing of an embedded computer from the machine in which it is embedded.

  Limited observability and controllability→*No keyboard and screens,* in real-time applications we may not be able to easily stop the system to see what is going on inside and to affect the system's operation.

4.3) Restricted development environments:

- We generally compile code on one type of machine, such as a PC, and download it onto the embedded system.
- To debug the code, we must usually rely on programs that run on the PC or workstation and then look inside the embedded system.

Performance in Embedded Computing

- Embedded system designers have to set their goal —their program must meet its ***deadline.***

Performance Analysis

1. CPU: The CPU clearly influences the behavior of the program, particularly when the CPU is a pipelined processor with a cache.
2. Platform: The platform includes the bus and I/O devices. The platform components that surround the CPU are responsible for feeding the CPU and can dramatically affect its performance.
3. Program: Programs are very large and the CPU sees only a small window of the program at a time. We must consider the structure of the entire program to determine its overall behavior.
4. Task: We generally run several programs simultaneously on a CPU, creating a multitasking system. The tasks interact with each other in ways that have profound implications for performance.
5. Multiprocessor: Many embedded systems have more than one processor—they may include multiple programmable CPUs as well as accelerators. Once again, the interaction between these processors adds yet more complexity to the analysis of overall system performance.

2)EMBEDDED SYSTEM DESIGN PROCESS

Design process has two objectives as follows.

1. It will give us an introduction to the various steps in embedded system design.
2. Design methodology

I. Design to ensure that we have done everything we need to do, such as optimizing performance or performing functional tests.

II. It allows us to develop computer-aided design tools.

III. A design methodology makes it much easier for members of a design team to communicate.

Levels of abstraction in the design process.

1)Requirements

•It can be classified in to functional or nonfunctional

1.1)Functional Requirements

•Gather an informal description from the customers.

•Refine the requirements into a specification that contains enough information to design the system architecture.

•Ex:Sample Requirements form

•Name→Giving a name to the project

Purpose→Brief one- or two-line description of what the system is supposed to do.

•Inputs& Outputs →Analog electronic signals? Digital data? Mechanical inputs?

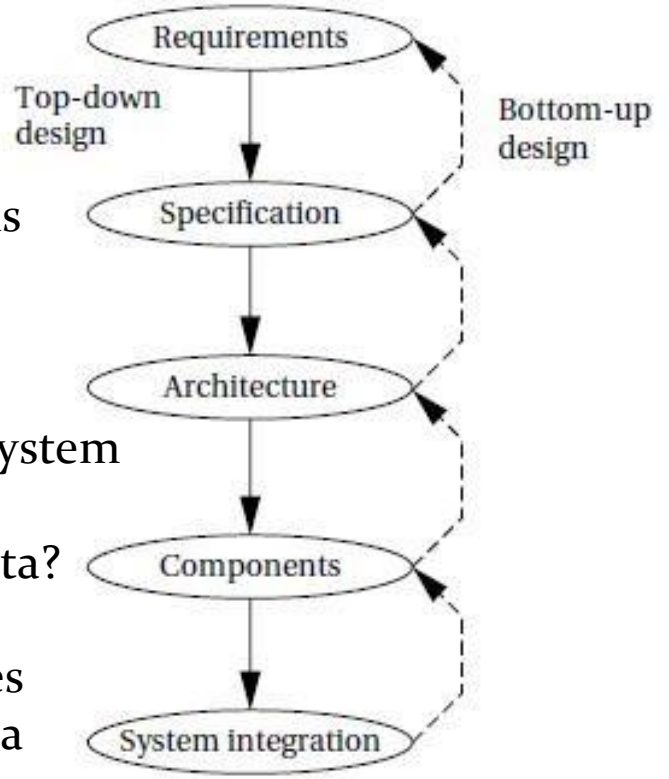
•Functions→ detailed description of what the system does

Performance→ computations must be performed within a certain time frame

•Manufacturing cost→ cost of the hardware components.

Power→ how much power the system can consume

Physical size and weight→ indication of the physical size of the system



- 1.2) Non-Functional Requirements
- Performance→ depends upon approximate time to perform a user-level function and also operation must be completed within deadline.
- Cost→Manufacturing cost includes the cost of components and assembly.
- Nonrecurring engineering (NRE) costs include the personnel and other costs of designing the
- system
- Physical Size and Weight→The final system can vary depending upon the application.
- Power Consumption→Power can be specified in the requirements stage in terms of battery life.

2)SPECIFICATION

- The specification must be carefully written so that it accurately reflects the customer's requirements.
- It can be clearly followed during design.

3) Architecture Design

- The architecture is a plan for the overall structure of the system.
- It is in the form block diagram that shows a major operation and data flow.

4) Designing Hardware and Software Components

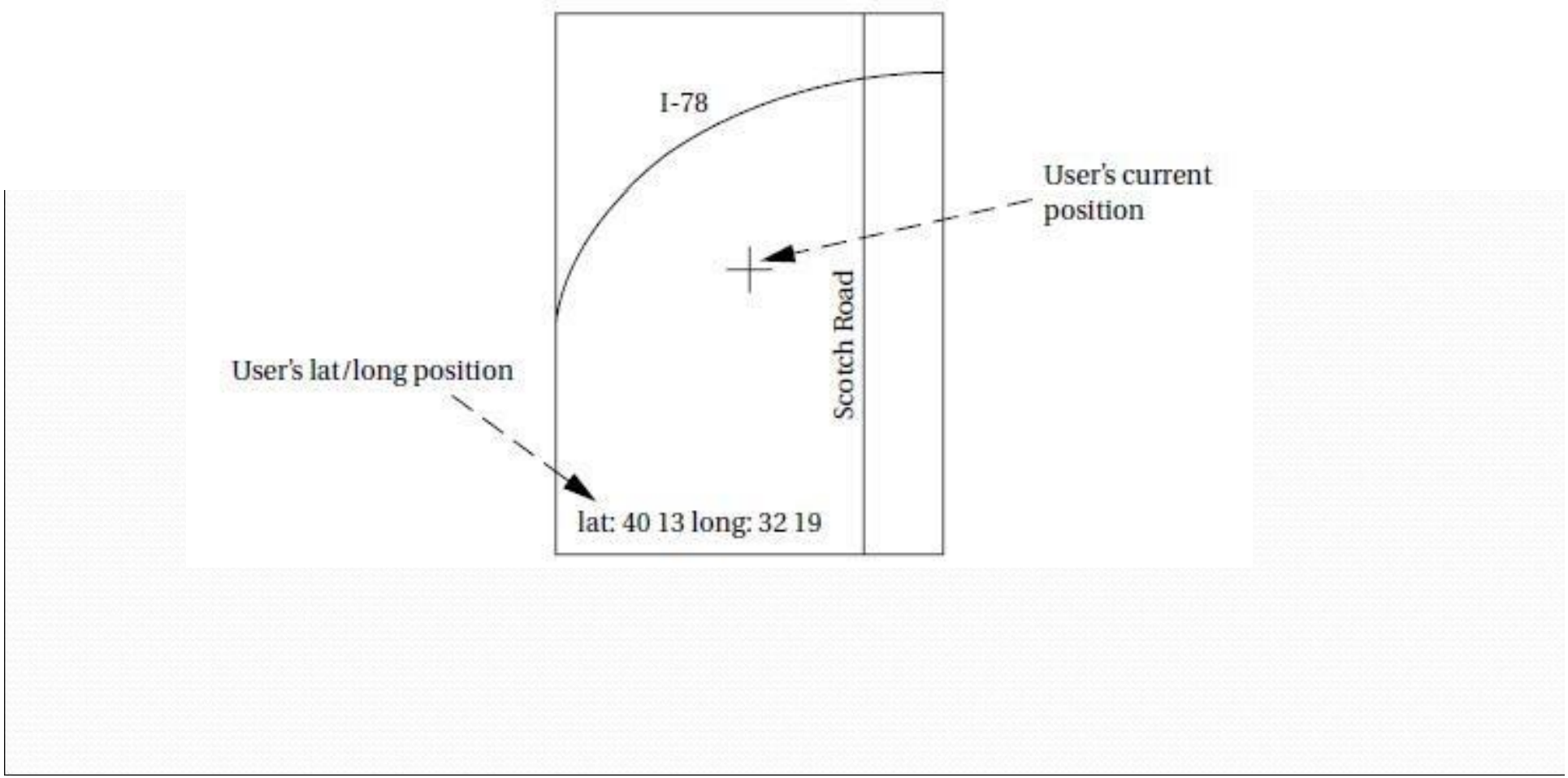
- The architectural description tells us what components we need include both hardware—FPGAs, boards & software modules

5)System Integration

- Only after the components are built, putting them together and seeing a working system.
- Bugs are found during system integration, and good planning can help us find the bugs quickly.

Embedded system Design Example

- GPS moving map



Design Process Steps

1. 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.
- The moving map obtains its position from the GPS, a satellite-based navigation system.

Name	GPS moving map
Purpose	Consumer-grade moving map for driving use
Inputs	Power button, two control buttons
Outputs	Back-lit LCD display 400 600
Functions	Uses 5-receiver GPS system; three user-selectable resolutions;always displays current latitude and longitude
Performance	Updates screen within 0.25 seconds upon movement
Manufacturing cost	$30
Power	100mW
Physical size and weight	No more than 2"X 6, " *12 ounces*

Design Process Steps

2) Functionality→This system is designed for highway driving and similar uses. The system should show major roads and other landmarks available in standard topographic databases.

3) User interface→The screen should have at least 400X600 pixel resolution. The device should be controlled by no more than 3 buttons.
→A menu system should pop up on the screen when buttons are pressed to allow the user to make selections to control the system.

4) Performance→ The map should scroll smoothly.
→Upon power-up, a display should take no more than 1sec to appear.
→The system should be able to verify its position and display the current map within 15 s.

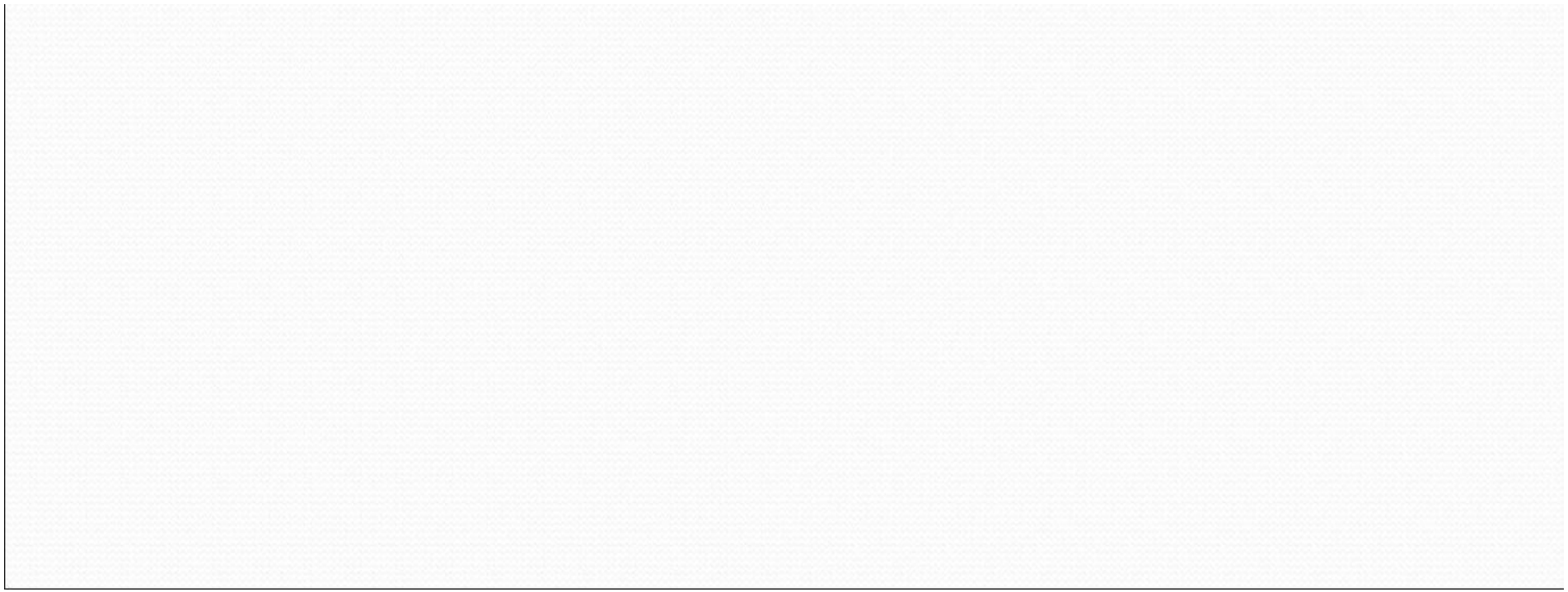
5) Cost→ The selling cost of the unit should be no more than $100.

6) Physical size and weight→The device should fit comfortably in the palm of the hand.

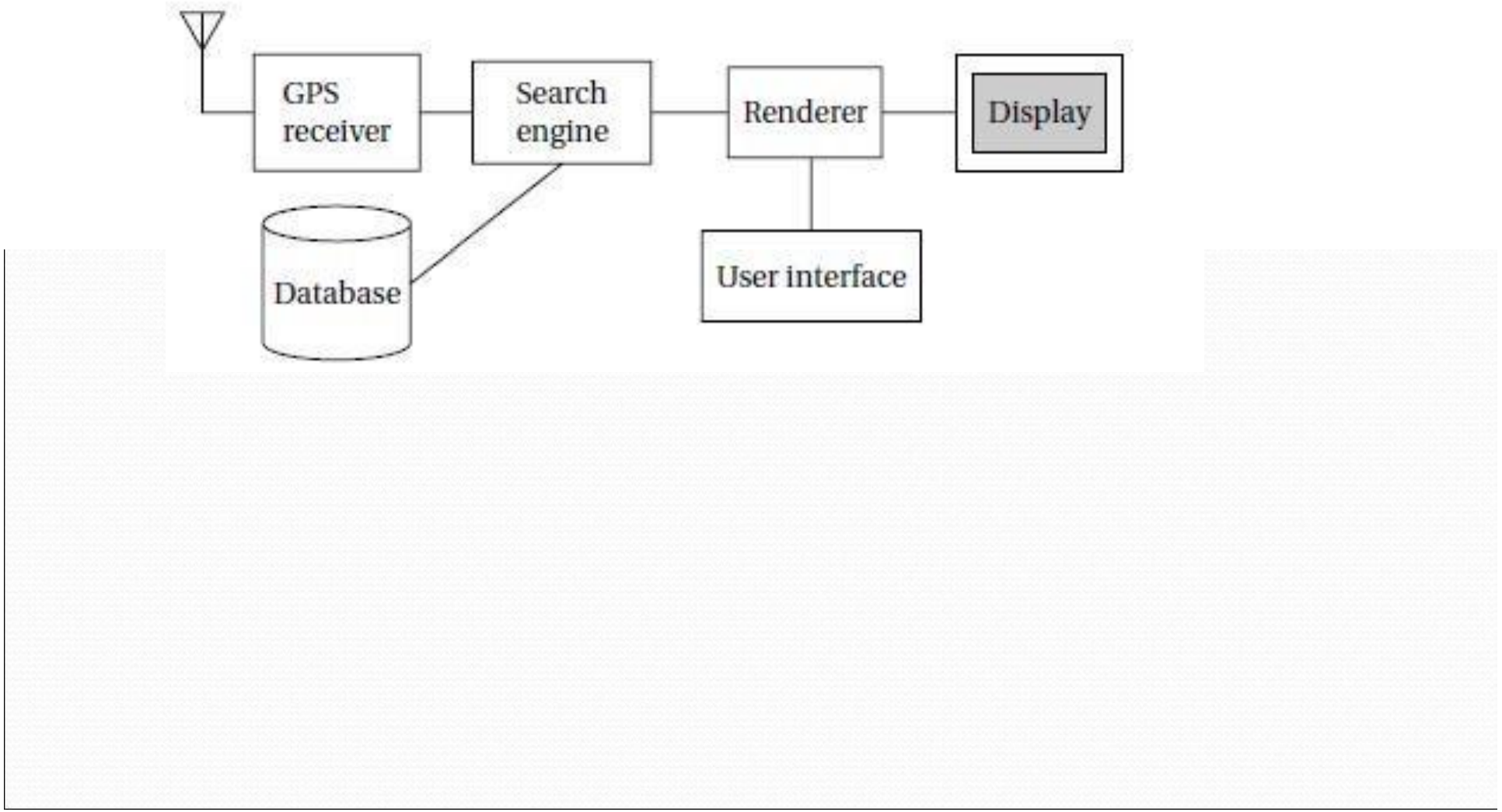
7) Power consumption→ The device run for at least 8 hrs on 4 AA batteries.

8) specification

1. Data received from the GPS satellite constellation.
2. Map data.
3. User interface.
4. Operations that must be performed to satisfy customer requests.
5. Background actions required to keep the system running, such as operating the GPS receiver.

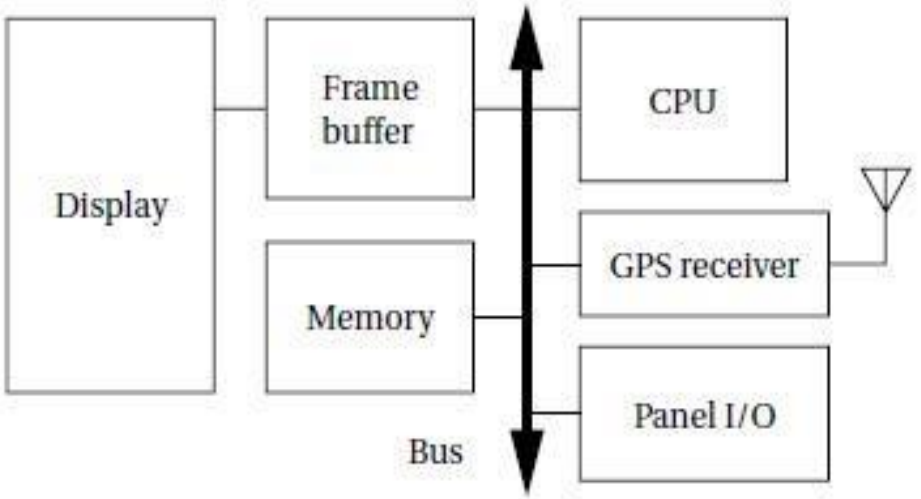


Block Diagram



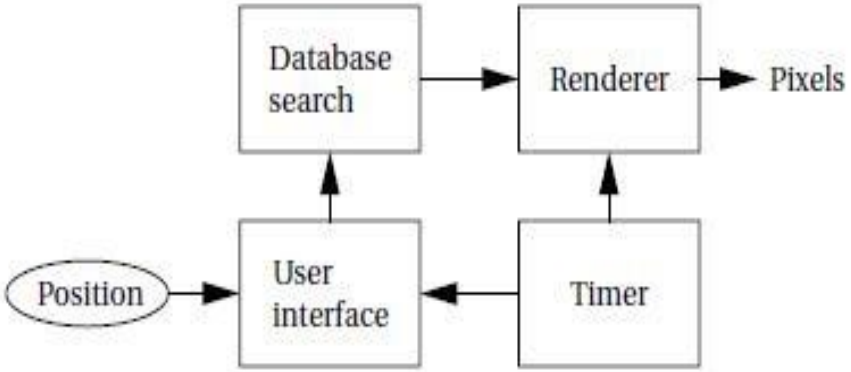
Hardware architecture

•one central CPU surrounded by memory and I/O devices.
• It used two memories: a frame buffer for the pixels to be displayed and a separate program/data memory for general use by the CPU.



Software architecture

•Timer to control when we read the buttons on the user interface and render data onto the screen.
•Units in the software block diagram will be executed in the hardware block diagram and when operations will be performed in time.



3)FORMALISM FOR SYSTEM DESIGN

- UML(Unified Modeling Language) is an object-oriented modeling language→ used to capture all these design tasks.
- It encourages the design to be described as a number of interacting objects, rather than blocks of code.
- objects will correspond to real pieces of software or hardware in the system.
- It allows a system to be described in a way that closely models real-world objects and their interactions.

Classification of descriptor

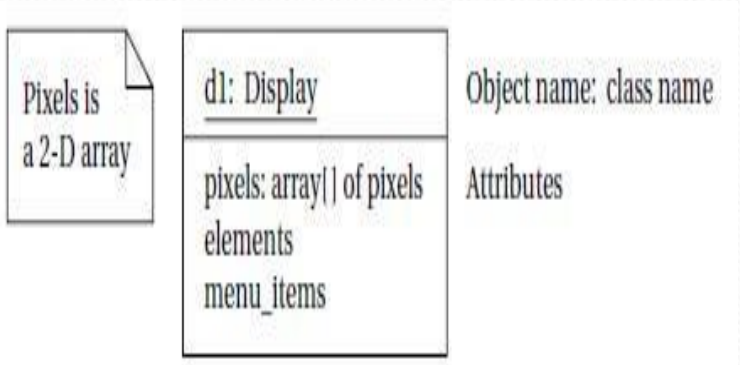
3.1)Structural Description

3.2)Behavioral Description

Structural Description

- It gives basic components of the system and designers can learn how to describe these components in terms of object.

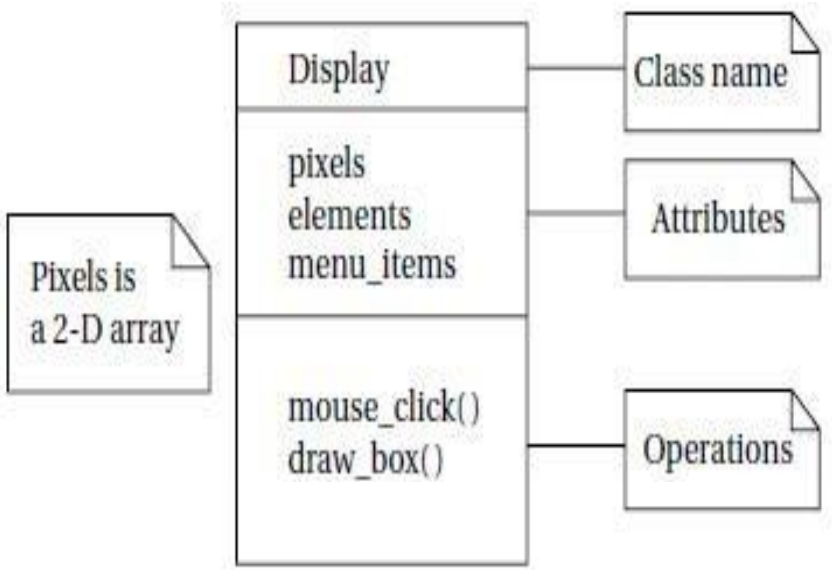
  OBJECT in UML NOTATION
- *An object* includes a set of attributes that define its internal state.
- An object describing a display (CRT screen) is shown in UML notation in Figure.
- The object has a **unique name**, and a member of a **class.**
- The name is underlined to show that this is a description of an object and not of a class.
- The text in the folded-corner page icon is a **note.**



An object in UML notation

CLASS IN UML NOTATION

- All objects derived from the same class have the same characteristics, but attributes may have different values.
- It also defines the **operations** that determine how the object interacts with the rest of the world.
- It defines both the interface for a particular type of object and that object's implementation.

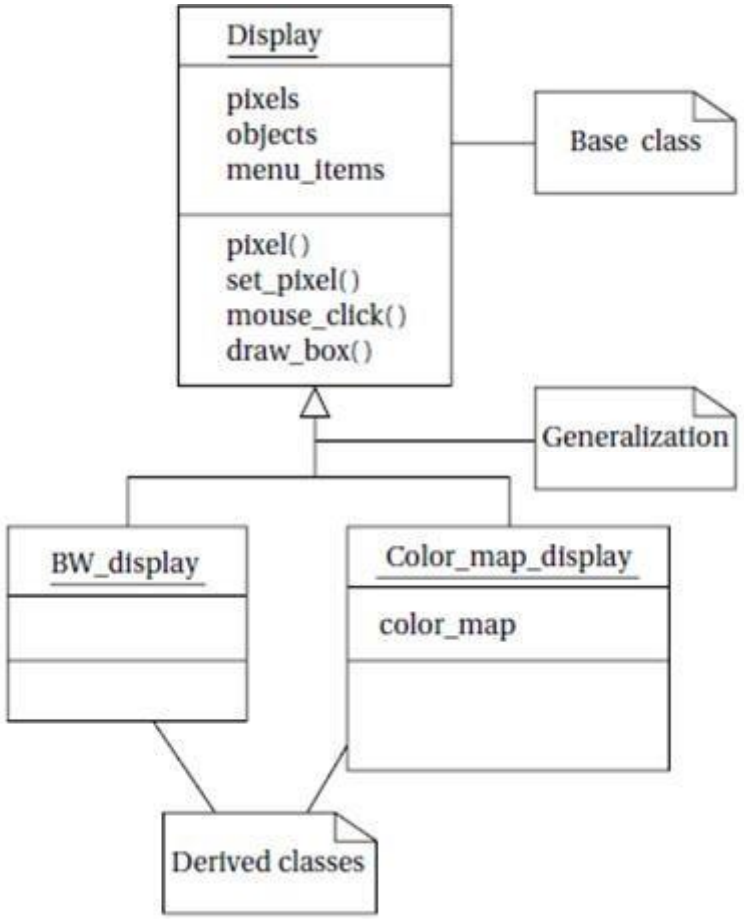


Relationships between objects and classes

1. Association →occurs between objects that communicate with each other but have no ownership relationship between them.
2. Aggregation→ describes a complex object made of smaller objects.
3. Composition →It is a type of aggregation in which the owner does not allow access to the component objects.
4. Generalization →allows us to define one class in terms of another.

Derived classes as a form of generalization in UML

•A derived class is defined to include all the attributes of its base class.
• Display is the base class and BW display and color map display are the two derived classes.
•BW display represents black and white display.



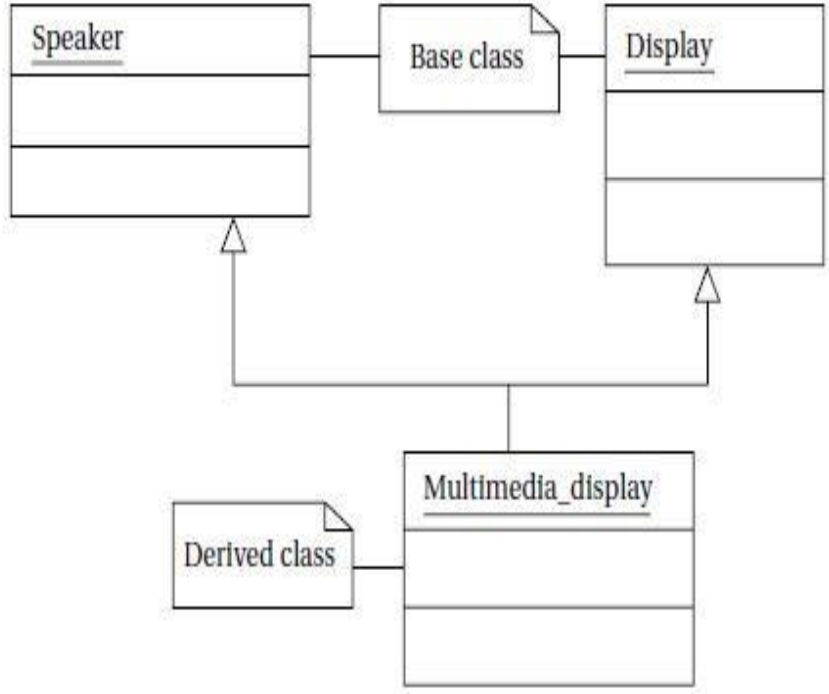
ultiple inheritance in UML

•UML allows to define multiple inheritance, in which a class is derived from more than one base class.

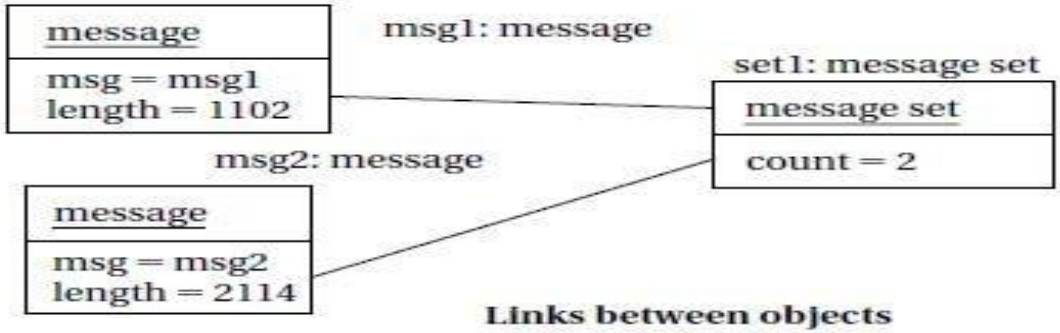
•Multimedia display class by combining the Display class with a Speaker class for sound.

•The derived class inherits all the attributes and operations of both its base classes, Display and Speaker.

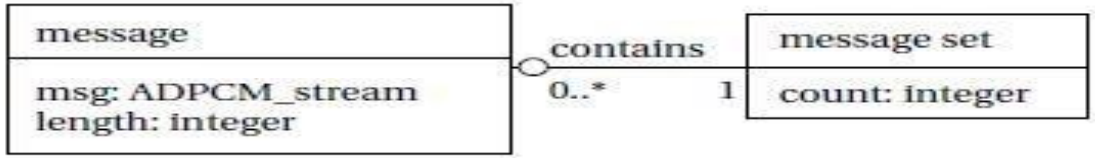
•



Links and Association



Links between objects

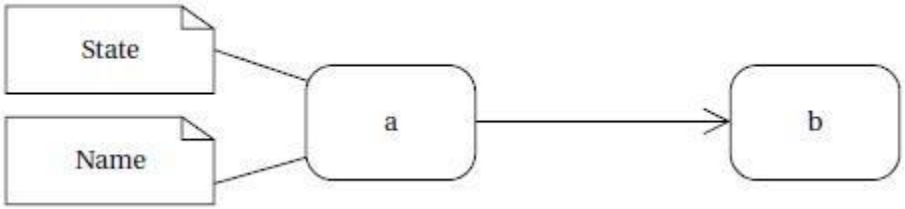


Association between classes

- A link describes a relationship between objects and association is to link as class is to object.
- Links used to make to stand associations capture type information about these links.
- The association is drawn as a line between the two labeled with the name of the association, namely, *contains*.

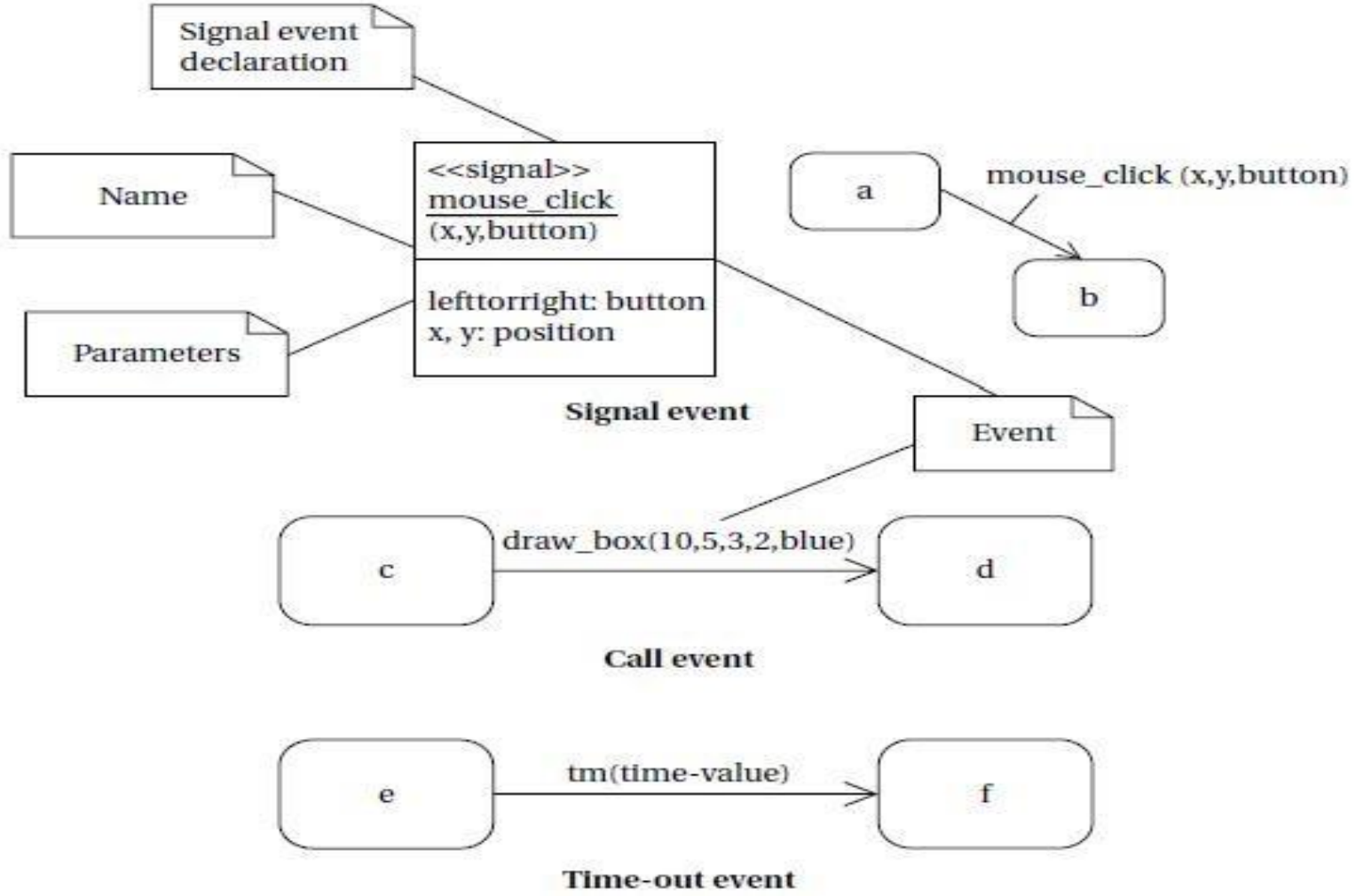
Behavioral Description

- Behavior of an operation is specified by a state machine.



- These state machines will not rely on the operation of a clock.
- Changes from one state to another are triggered by the occurrence of events.
- The event may generated from the outside or inside of the system.

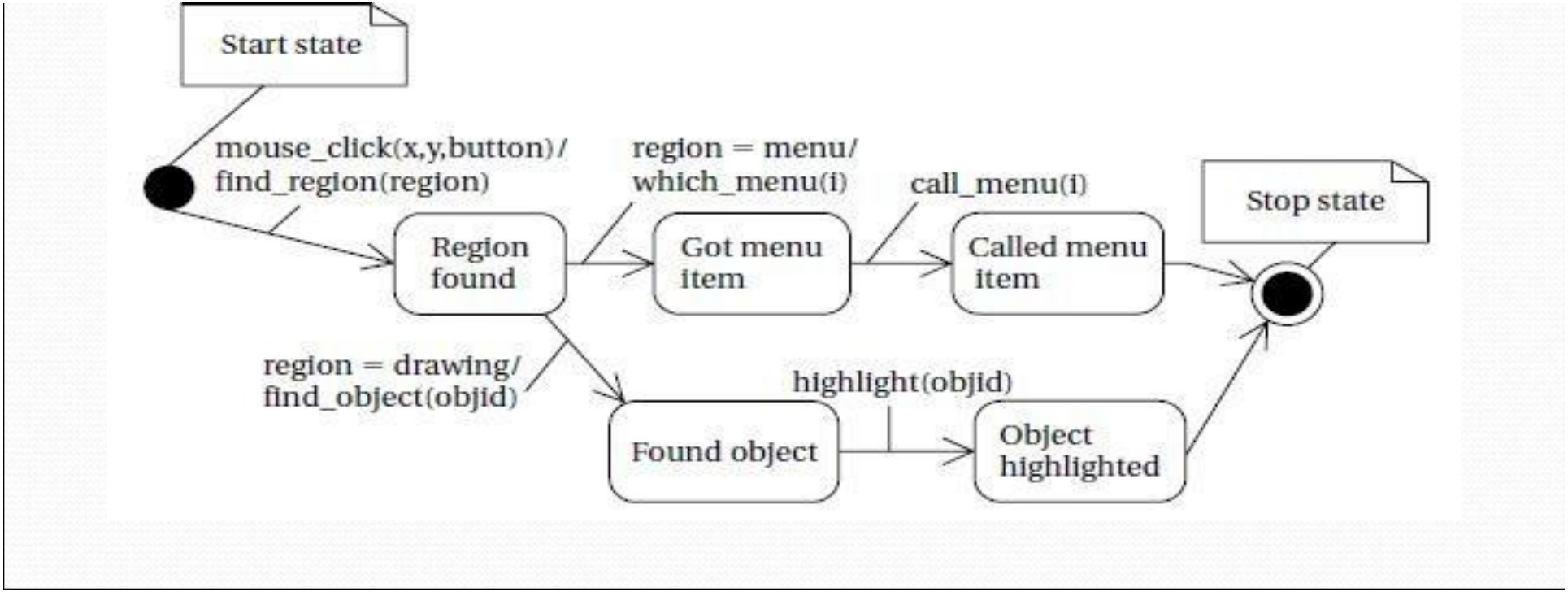
Signal, call, and time-out events in UML.



- Signal →is an asynchronous occurrence.
- It is defined in UML by an object that is labeled as a <<signal>>.
- Signal may have parameters that are passed to the signal's receiver.
- Call event→ follows the model of a procedure call in a programming language.
- Time-out event→ causes the machine to leave a state after a certain amount of time.
- The label tm(time-value) on the edge gives the amount of time after which the transition occurs.
- It is implemented with an external timer.

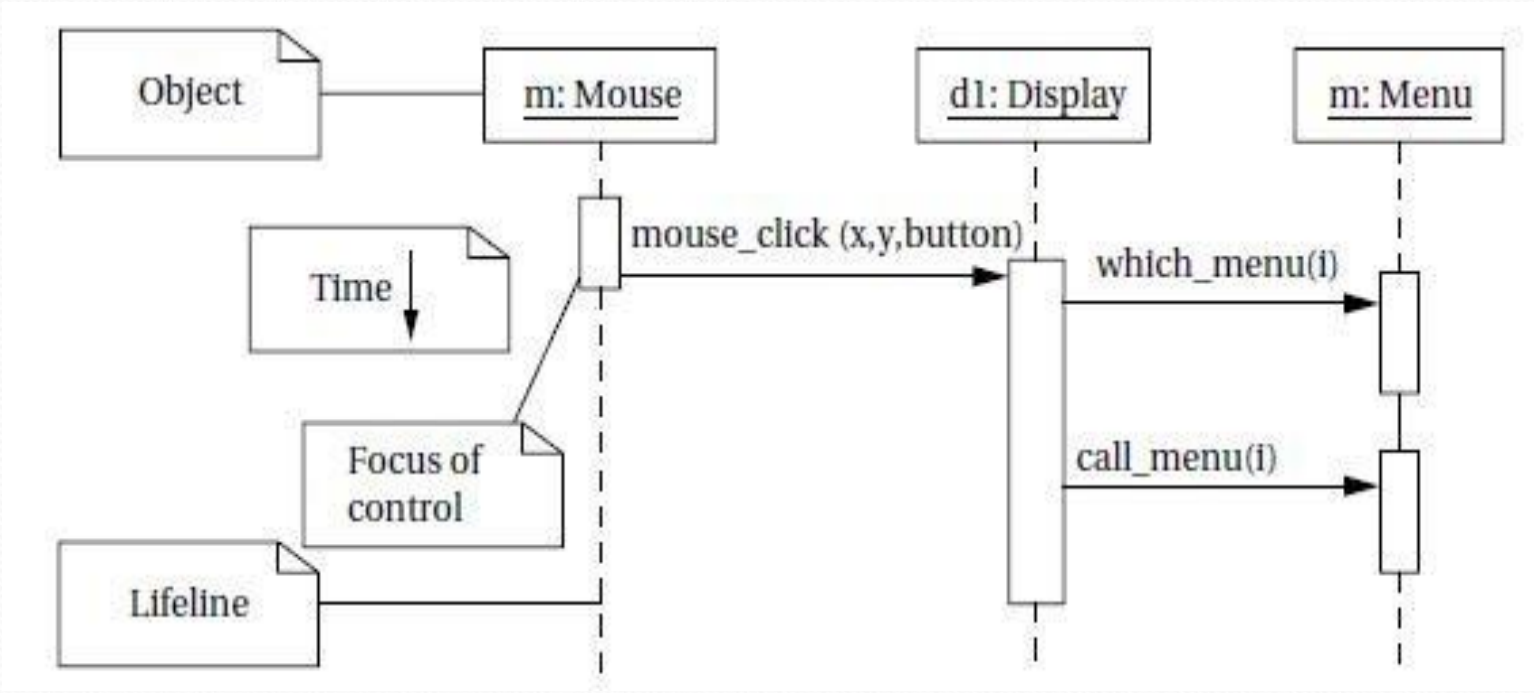
State Machine specification in UML

- The start and stop states are special states which organize the flow of the state machine.
- The states in the state machine represent different operations.
- Conditional transitions out of states based on inputs or results of some computation.
- An unconditional transition to the next state.



Sequence diagram in UML

- Sequence diagram is similar to a hardware timing diagram, although the time flows vertically in a sequence diagram, whereas time typically flows horizontally in a timing diagram.
- It is designed to show particular choice of events—it is not convenient for showing a number of mutually exclusive possibilities.



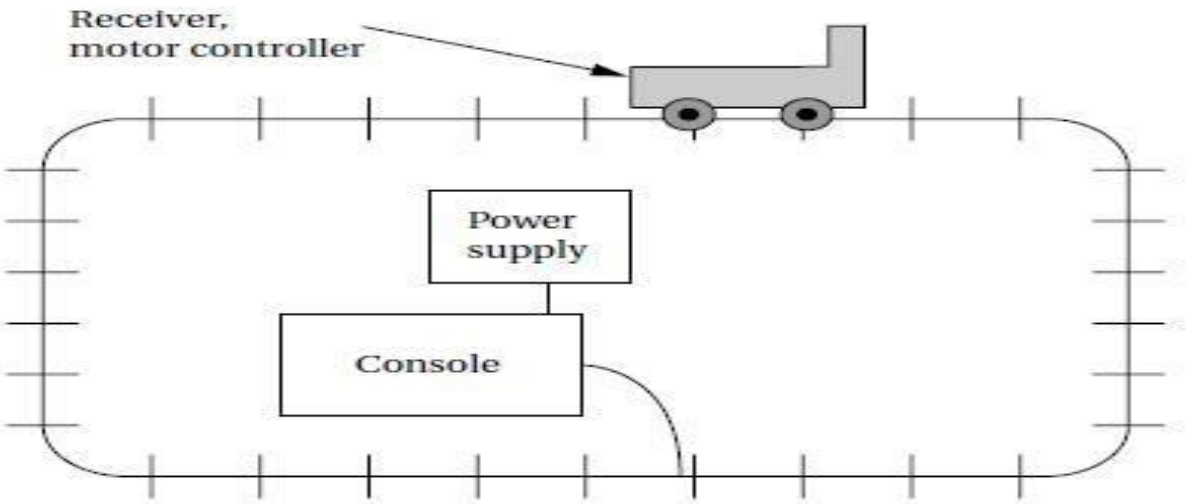
4) Design:Model Train Controller

- In order to learn how to use UML to model systems→ specify a simple system (Ex: model train controller)
- The user sends messages to the train with a control box attached to the tracks.
- The control box may have controls such as a throttle, emergency stop button, and so on.
- The train Rx its electrical power from the two rails of the track.

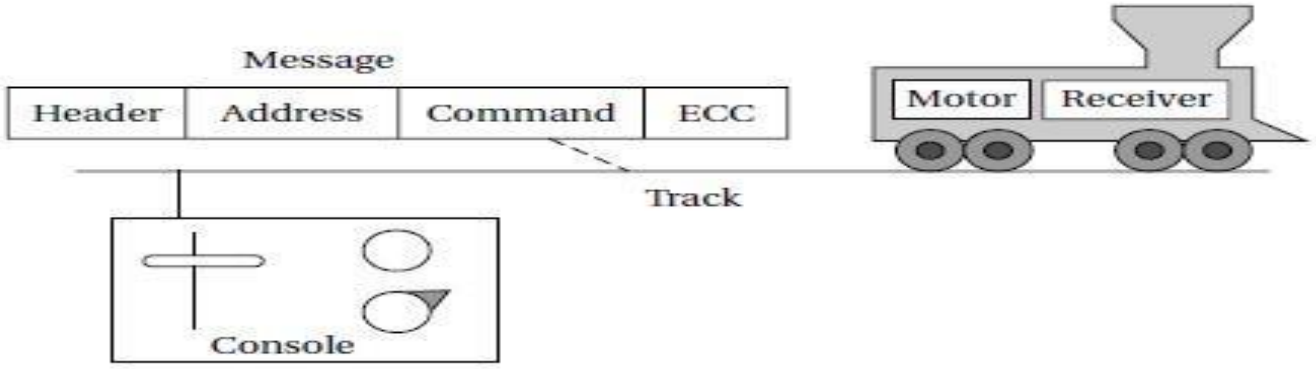
CONSOLE

- Each packet includes an address so that the console can control several trains on the same track.
- The packet also includes an error correction code (ECC) to guard against transmission errors.
- This is a one-way communication system—the model train cannot send commands back to the user.

Model Train Control system



System setup



Signaling the train

REQUIREMENTS

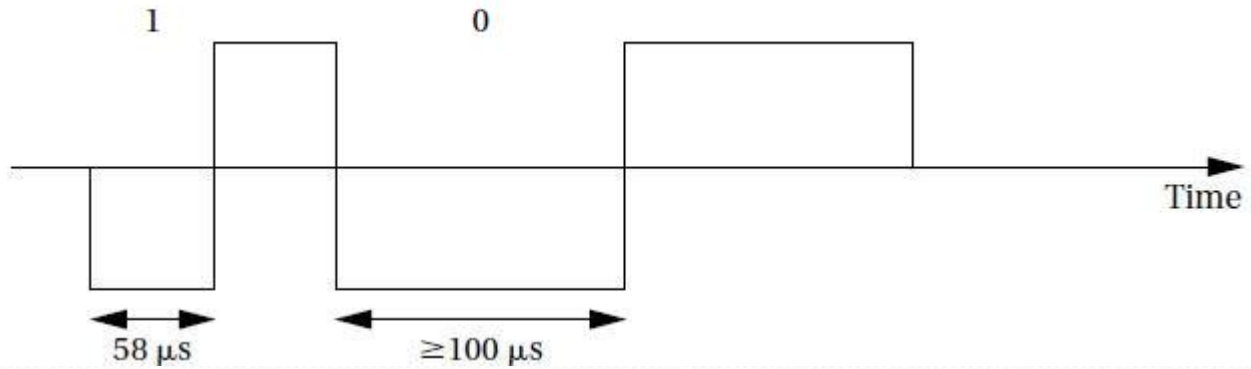
- The console shall be able to control up to eight trains on a single track.
- The speed of each train controllable by a throttle to at least 63 different levels in each direction (forward and reverse).
- There shall be an inertia control→ to adjust the speed of train.
- There shall be an emergency stop button.
- An error detection scheme will be used to transmit messages.

Requirements:Chart Format

Name	Model train controller
Purpose	Control speed of up to eight model trains
Inputs	Throttle, inertia setting, emergency stop, train number
Outputs	Train control signals
Functions respond	Set engine speed based upon inertia settings;
Performance	Can update train speed at least 10 times per second
Manufacturing cost	$50
Power	10W
Physical size and weight	Console should be comfortable for two hands,approximatesize of standard keyboard;
weight	2 pounds

Digital Command Control (DCC)

- Standard S-9.1→ how bits are encoded on the rails for transmission.
- Standard S-9.2→ defines the packets that carry information.
- The signal encoding system should not interfere with power transmission
- Data signal should not change the DC value of the rails.
- Bits are encoded in the time between transitions.
- Bit 0 is at least 100 s while bit 1 is nominally 58 s.



Packet Formation in DCC

- The basic packet format is given by $PSA(sD) + E$
- P → preamble, which is a sequence of at least 10 1 bits.
- S → packet start bit. It is a 0 bit.
- A →address is 8 bits long. The addresses 00000000, 1111110, and 1111111 are reserved.
- s→ data byte start bit, which, like the packet start bit, is a 0.
- D →data byte includes 8 bits. A data byte may contain an address, instruction, data, or error correction information.
- E →packet end bit, which is a 1 bit.

Baseline packet

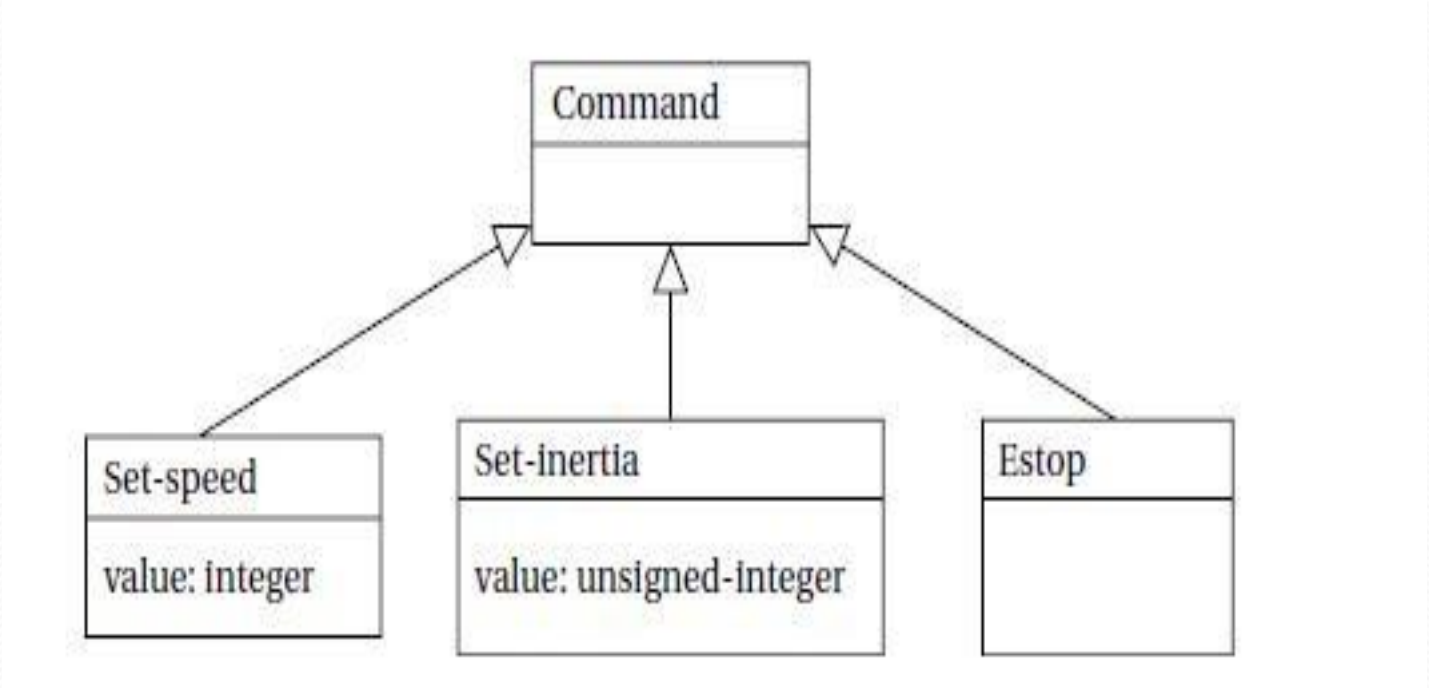
- The minimum packet that must be accepted by all DCC implementations.
- It has three data bytes.
- Address data byte→ gives the intended receiver of the packet
- Instruction data byte→ provides a basic instruction
- Error correction data byte→ is used to detect and correct transmission errors.

Date byte

- Bits 0–3→ provide a 4-bit speed value.
- Bit 4 →has an additional speed bit.
- Bit 5 →gives direction, with 1 for forward and 0 for reverse.
- Bits 6-7 are set at 01 → provides speed and direction.

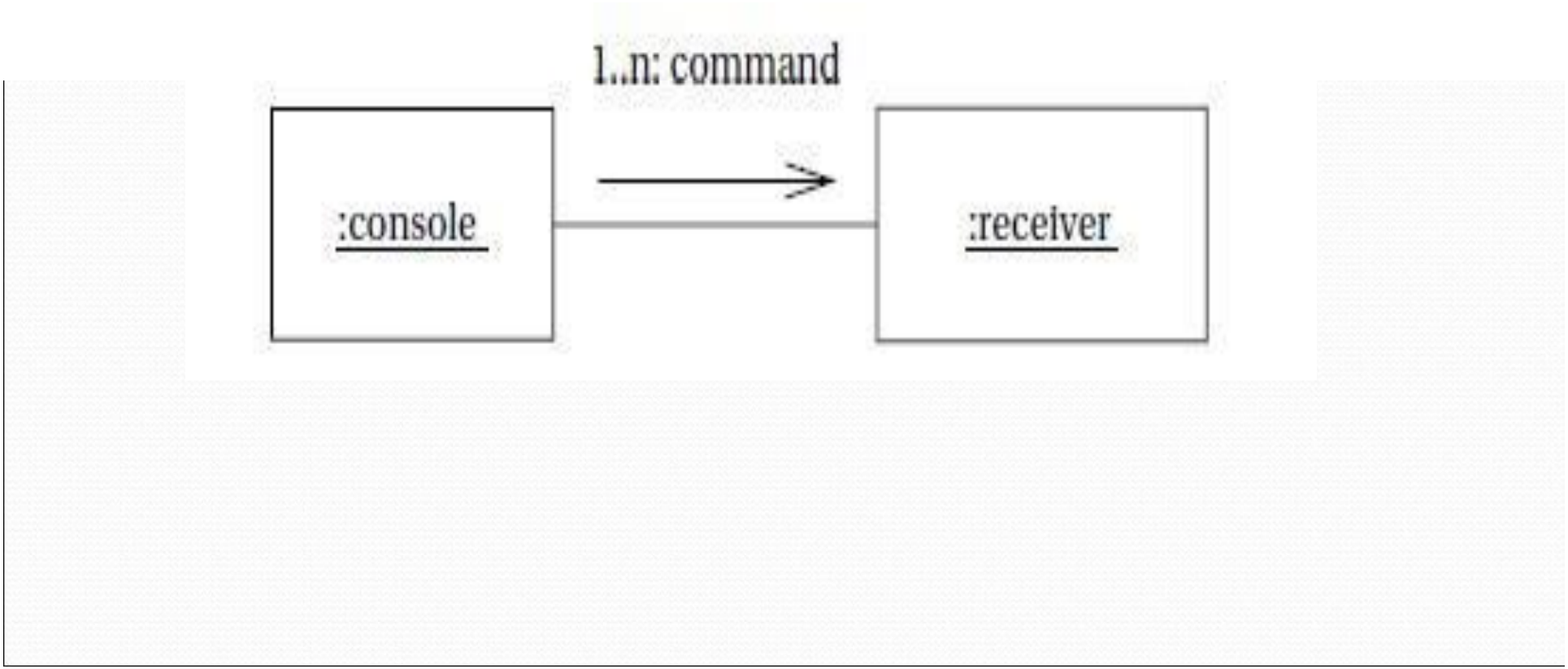
Conceptual Specification

- Conceptual specification allows us to understand the system a little better.
- A train control system turns commands into packets.
- A command comes from the command unit while a packet is transmitted over the rails.
- Commands and packets may not be generated in a 1-to-1 ratio

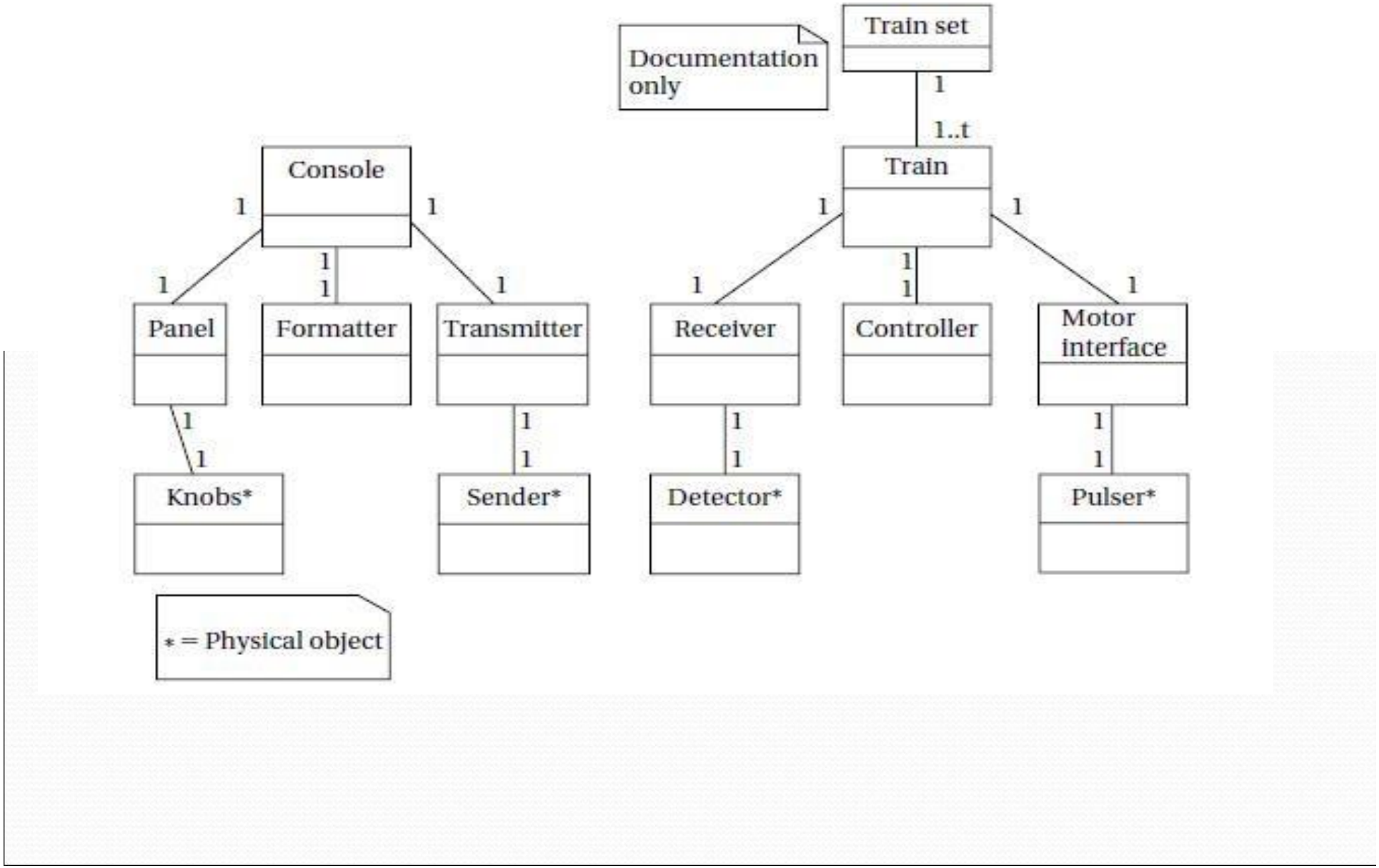


UML collaboration diagram for train controller system

- The command unit and receiver are each represented by objects.
- The command unit sends a sequence of packets to the train's receiver, as illustrated by the arrow messages as 1..n.
- Those messages are of course carried over the track.



UML class diagram for the train controller

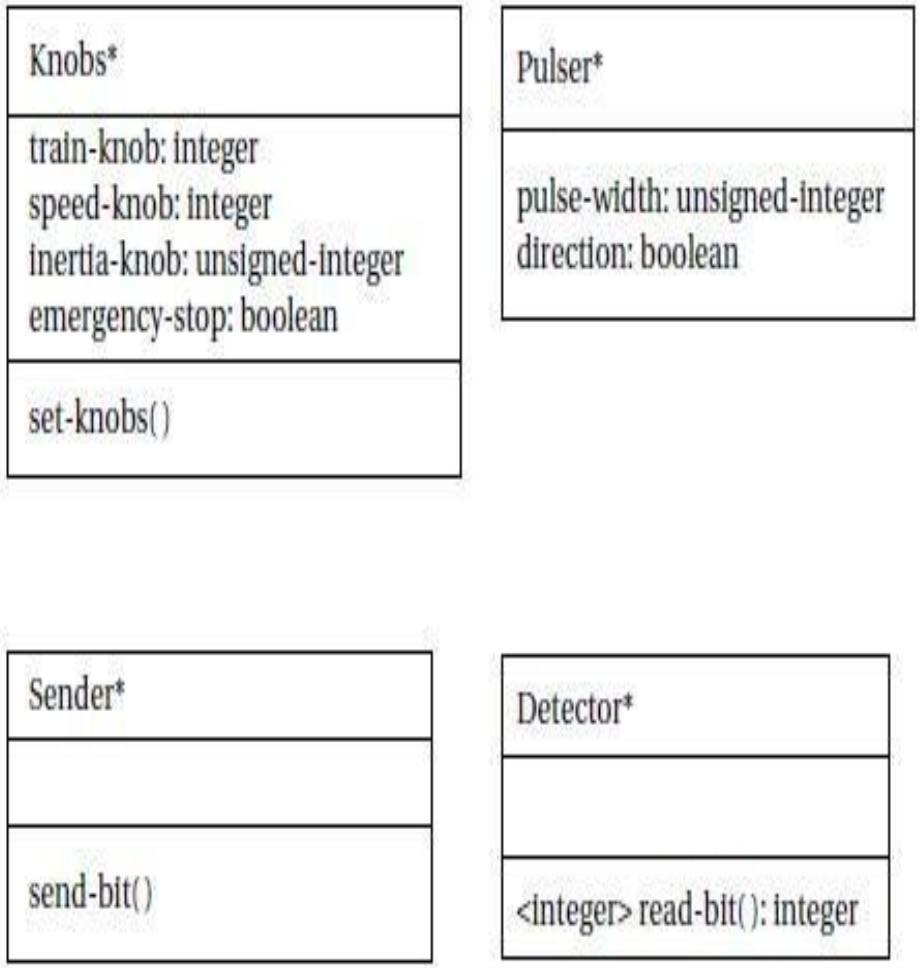


Basic characteristics of UML classes

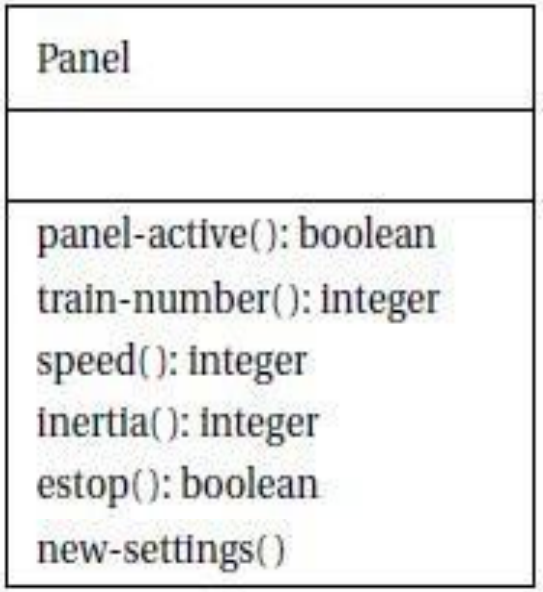
- *Console class→ describes the command unit's front panel, which contains* the analog knobs and hardware to interface to the digital parts of the system.
- *Formatter class→ includes behaviors that know how to read the panel* knobs and creates a bit stream for the required message.
- *Transmitter class →interfaces to analog electronics to send the message* along the track
- *Knobs*→ describes the actual analog knobs, buttons, and levers on the control* panel.
- *Sender*→ describes the analog electronics that send bits along the track.*
- *Receiver class→ knows how to turn the analog signals on the track into* digital form.
- *Controller class→ includes behaviors that interpret the commands and* figures out how to control the motor.
- *Motor interface class→ defines how to generate the analog signals required* to control the motor.
- *Detector* →detects analog signals on the track and converts them into digital* form.
- *Pulser* →turns digital commands into the analog signals required to control the* motor speed.

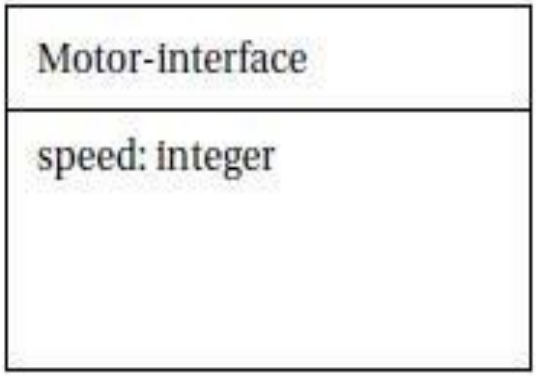
Detailed Specification

•The Panel has three knobs
• train number (which train is currently being controlled).
•speed (which can be positive or negative), and inertia.
•It also has one button for emergency-stop.
•When we change the train number setting, to reset the other controls to the proper values for that train.
• so that the previous train's control settings are not used to change the current train's settings.



Class diagram for panel

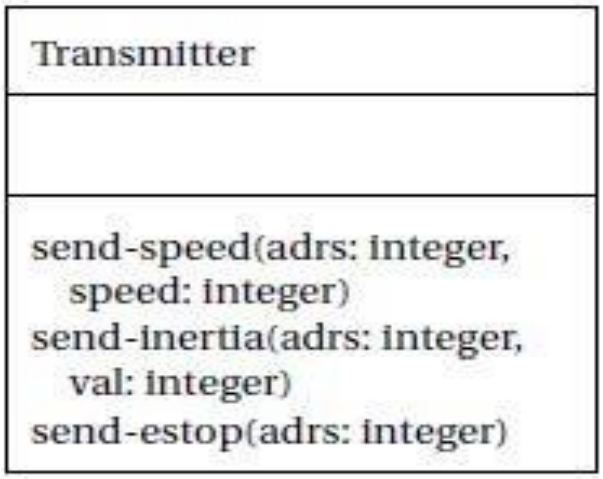


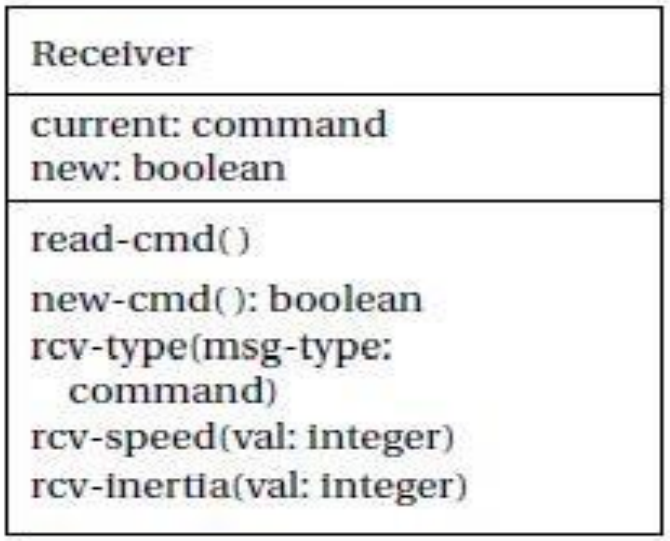


- The *Panel class* defines a behavior for each of the controls on the panel.
- The *new-settings behavior uses the set-knobs behavior of the Knobs**
- Change the knobs settings whenever the train number setting is changed.
- The *Motor-interface defines an attribute for speed that can be set by other classes.*

.

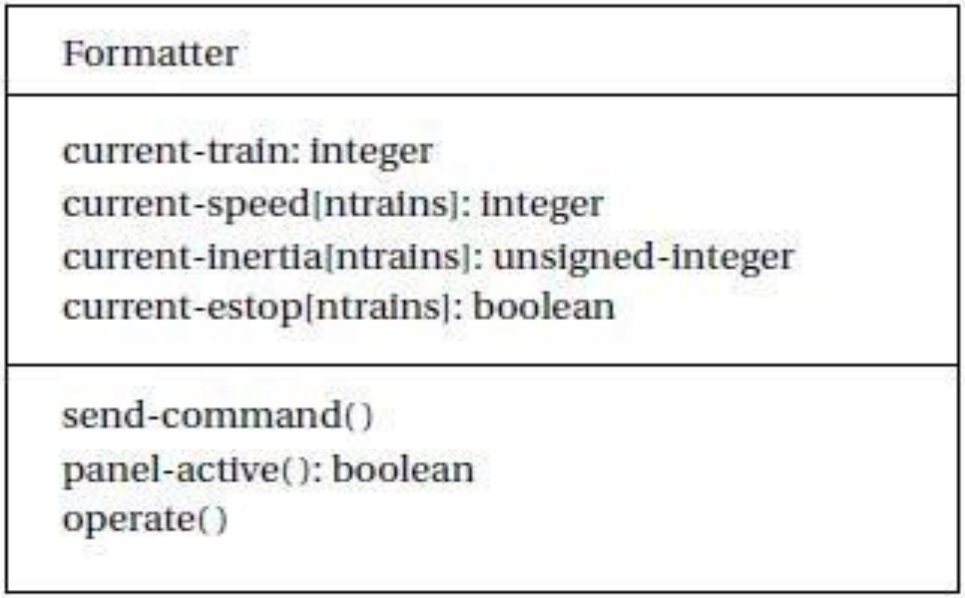
Class diagram for the Transmitter and Receiver





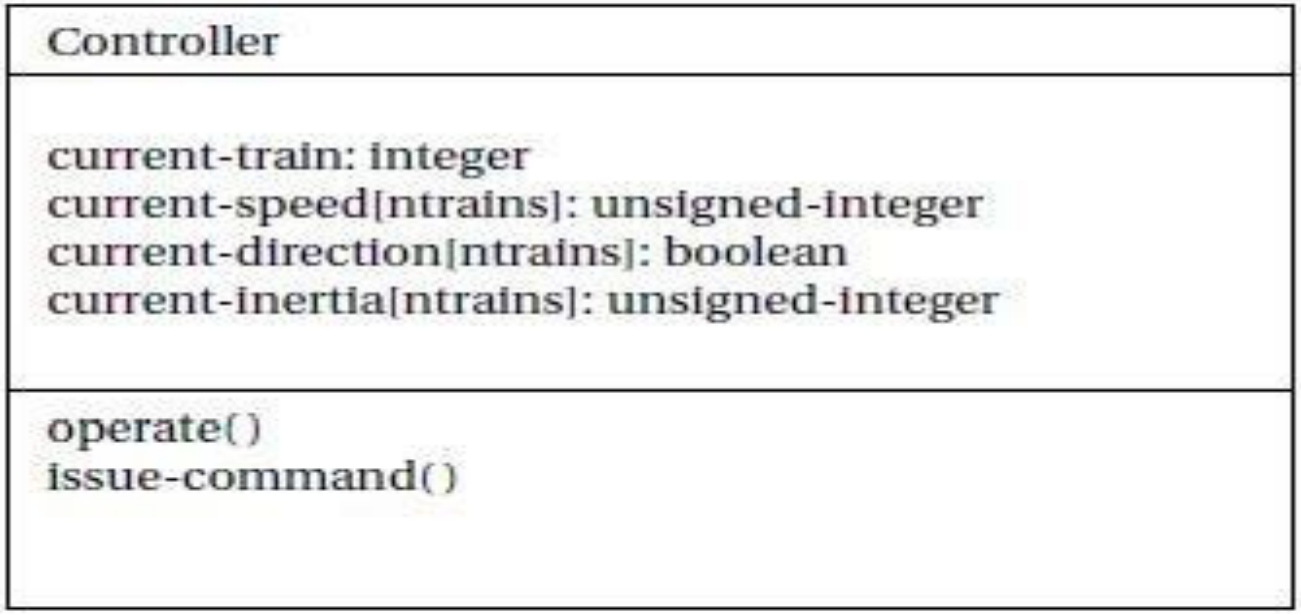
- They provide the software interface to the physical devices that send and receive bits along the track.
- The Transmitter provides a behavior message that can be sent
- The Receiver class provides a read-cmd behavior to read a message off the tracks.

Class diagram for Formatter



- The formatter holds the current control settings for all of the trains.
- The *send-command* serves as the interface to the transmitter.
- The *operate function performs the* basic actions for the object.
- The *panel-active behavior* returns true whenever the panel's values do not correspond to the current values

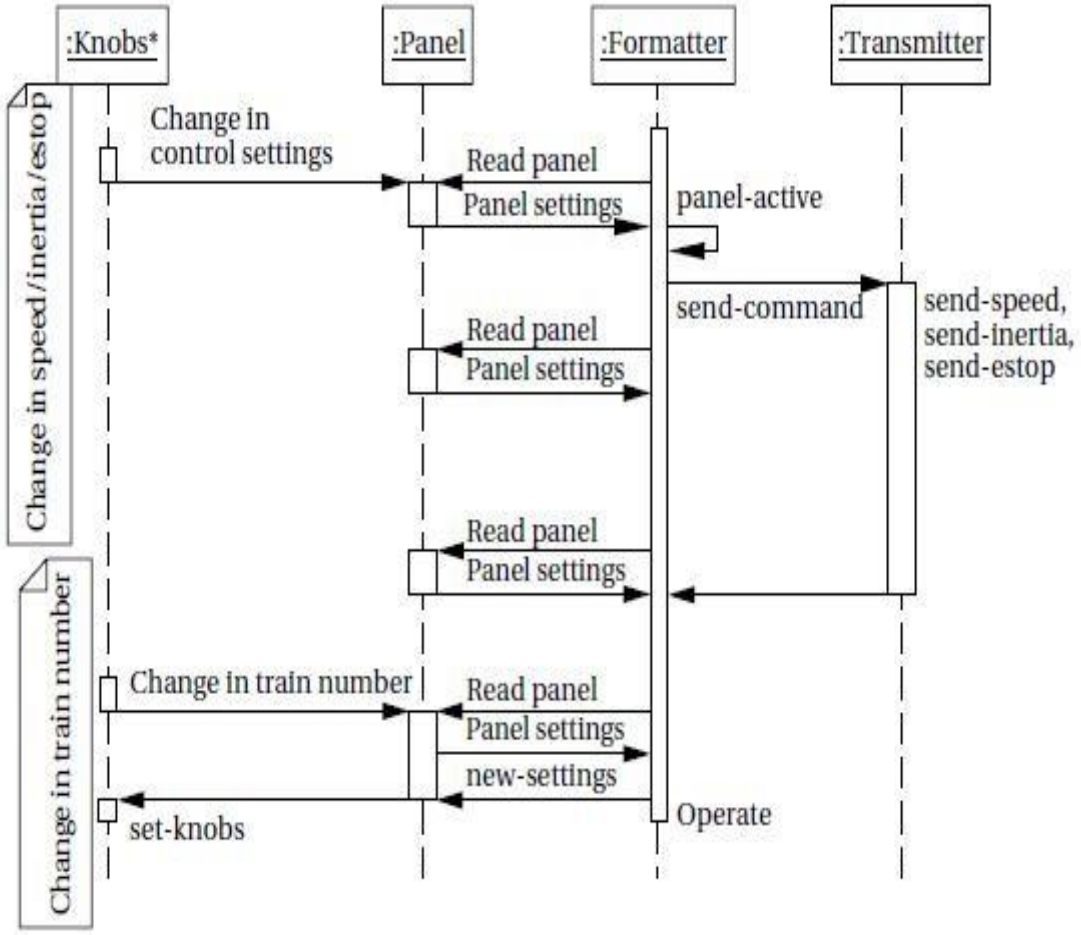
Class diagram for Controller



- The *Controller's operate behavior must execute* several behaviors to determine the nature of the message.
- Once the speed command has been parsed, it must send a sequence of commands to the motor to smoothly change the train's speed.

Sequences diagram for transmitting a control input

- Sequence diagram specify the interface between more than one classes.
- Its detailed operations and what ways its going to operate



DESIGN METHODOLOGIES

- Design of Embedded system is not an easy task.
- The main goal of a design process is to create a product that does something useful.
- Typical specifications for a product are functionality , manufacturing cost, performance and power consumption.

Design process has several important goals as follows

Time-to-market

- Customers always want new features.
- The product that comes out first can win the market, even setting customer preferences for future generations of the product.

Design cost

- Consumer products are very cost sensitive, and it is distinct from manufacturing cost.
- Design costs can dominate manufacturing costs.
- Design costs can also be important for high-volume consumer devices when time-to-market pressures cause teams to swell in size.

Quality

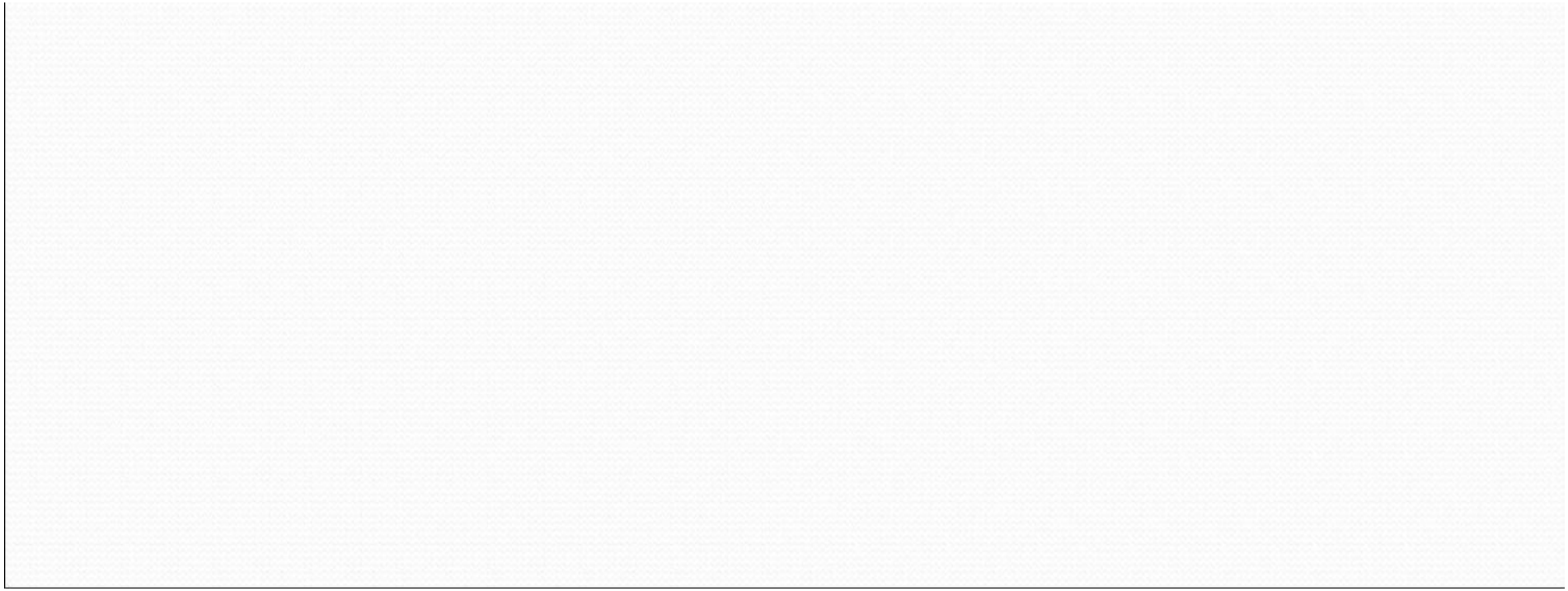
- Customers want their products fast and cheap.
- Correctness, reliability, and usability must be explicitly addressed from the beginning of the design job to obtain a high-quality product at the end

Design flows

- A design flow is a sequence of steps to be followed during a design.
- Some of the steps can be performed by tools and other steps can be performed by hand.

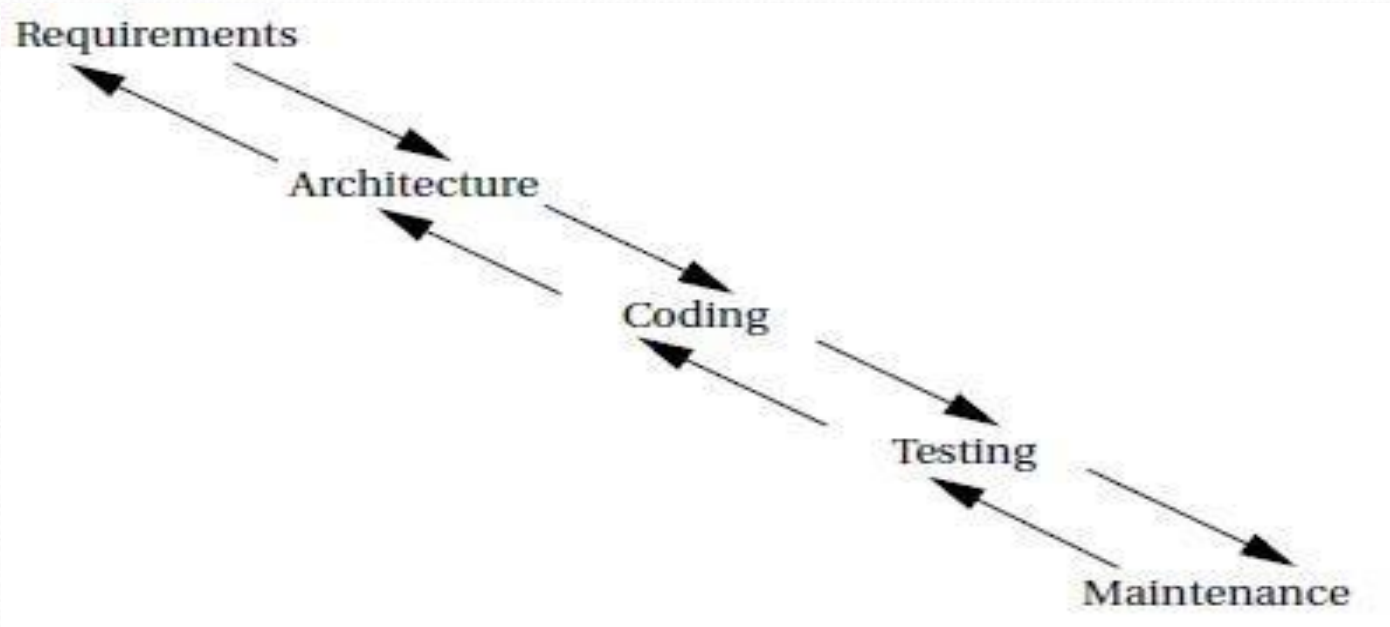
Types of Software development models

1. Waterfall model
2. Spiral model
3. Successive refinement development model
4. Hierarchical design model



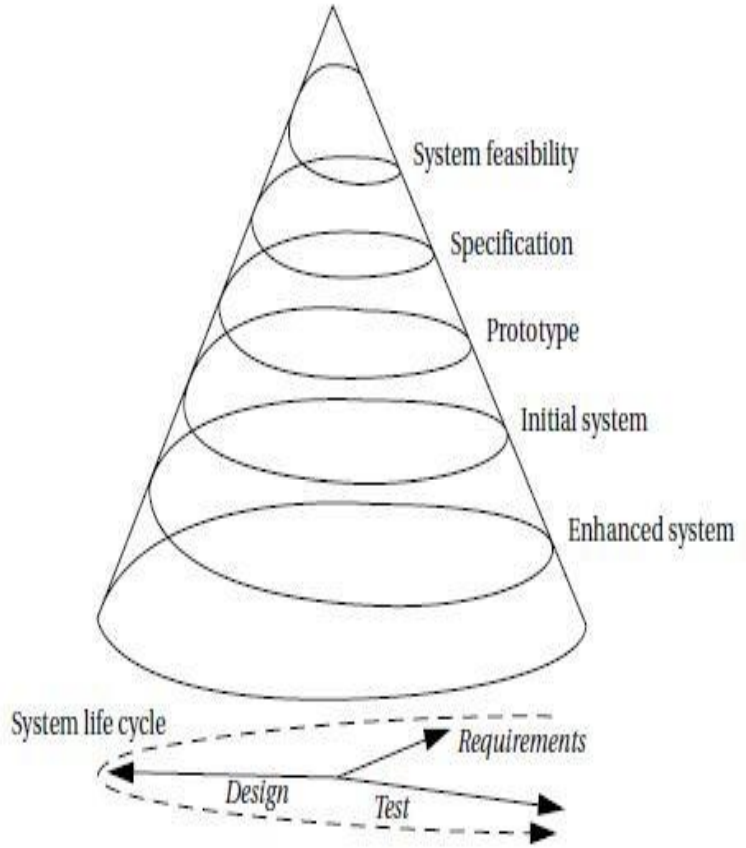
Waterfall model

- The waterfall development model consists of five major phases.
- Requirements analysis→ determines the basic characteristics of the system.
- Architecture design→It decomposes the functionality into major components
- Coding→It implements the pieces and integrates them.
- Testing→It detemines bugs.
- Maintenance→ It entails deployment in the field, bug fixes,and upgrades.
- The waterfall model makes work flow information from higher levels of abstraction to more detailed design steps.



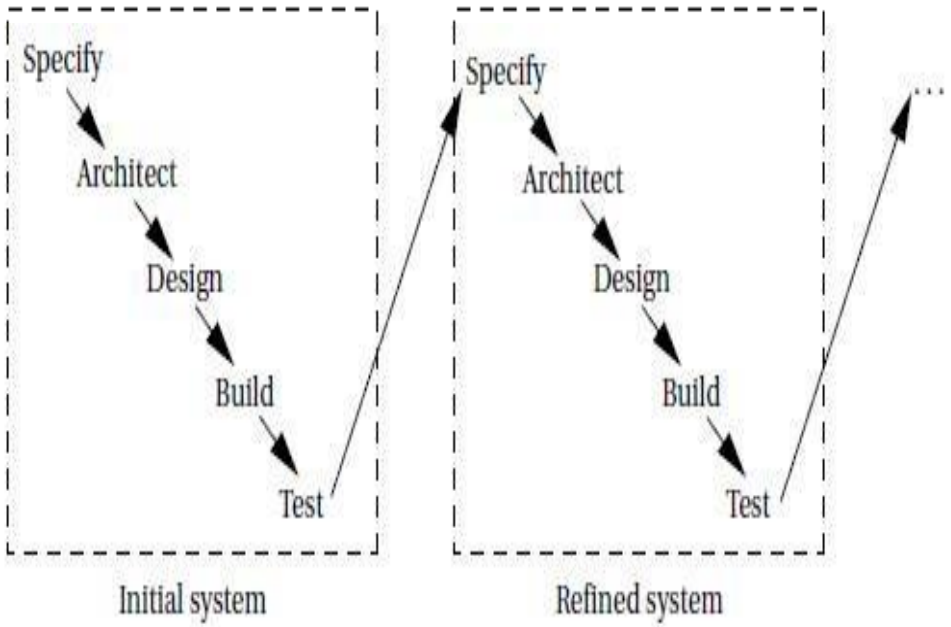
Spiral model

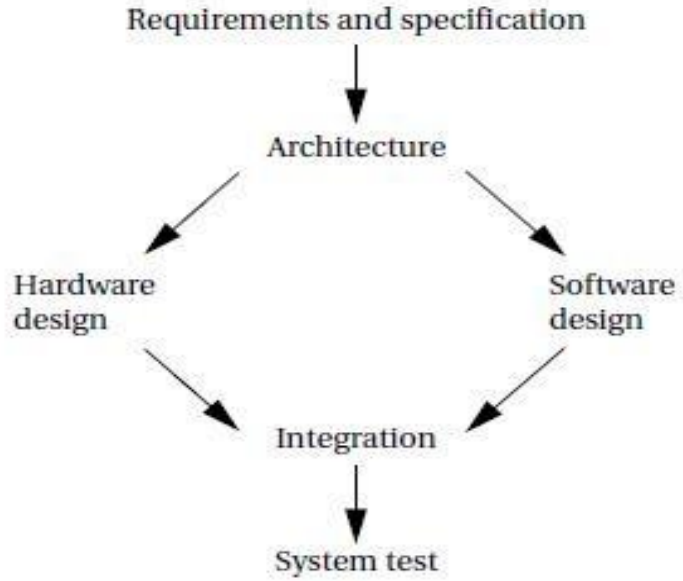
- The spiral model assumes that several versions of the system will be built.
- Each level of design, the designers go through requirements,construction,and testing phases.
- At later stages when more complete versions of the system are constructed.
- Each phase requires more work, widening the design spiral.
- The first cycles at the top of the spiral are very small and short.
- The final cycles at the spiral's bottom add detail learned from the earlier cycles of the spiral.
- The spiral model is more realistic than the waterfall model because multiple iterations needed to complete a design.
- But too many spirals may take long time required for design.

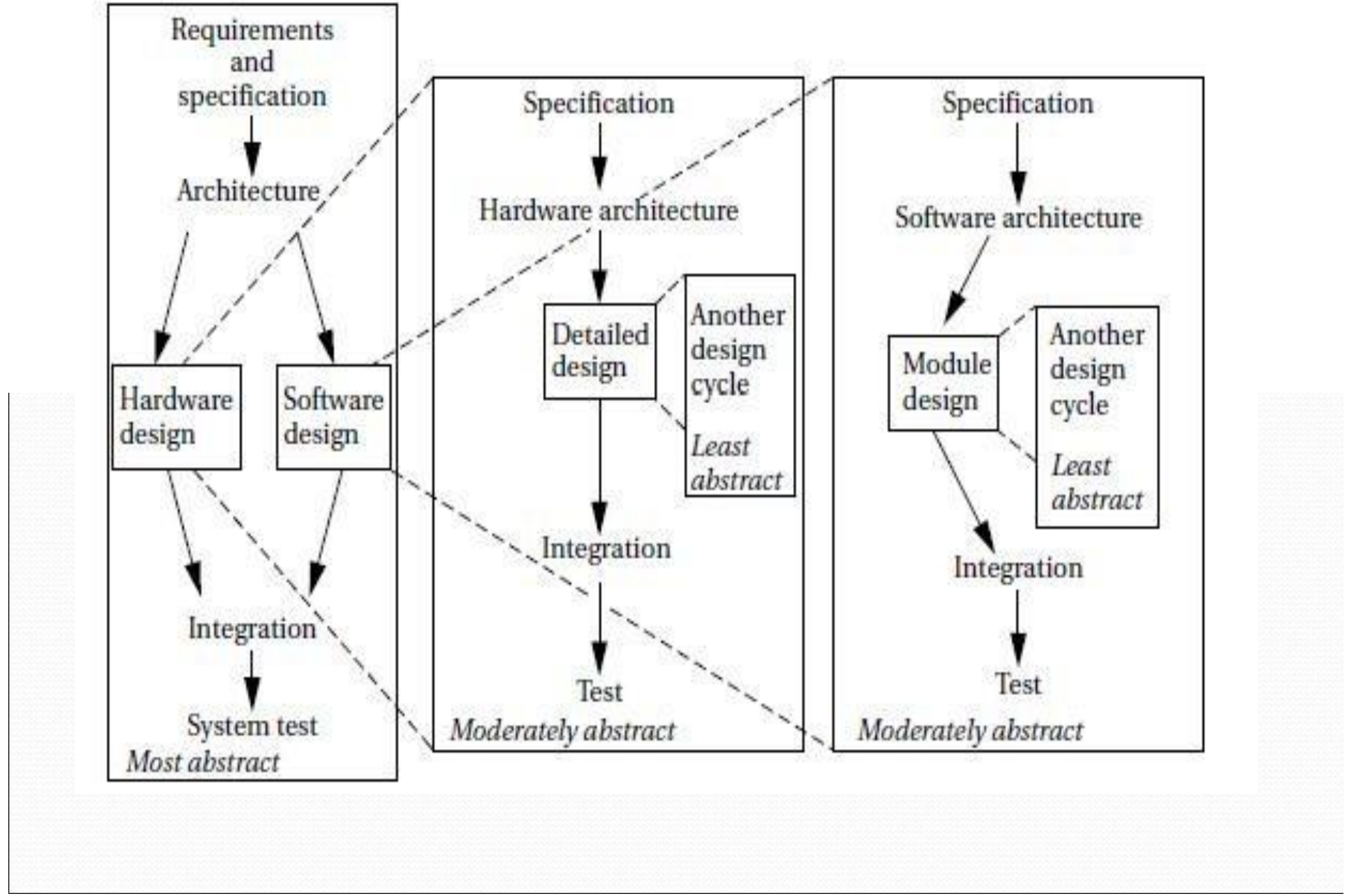


Successive refinement design model

- In this approach, the system is built several times.
- A first system is used as a rough prototype.
- Embedded computing systems are involved the design of hardware/software project.
- Front-end activities→ are specification and architecture and also includes hardware and software aspects.
- Back-end activities → includes integration and testing.
- Middle activities→includes hardware and software development.






Requirements and specification
Architecture
Hardware design
Software design
Integration
System test
Most abstract
Specification
Hardware architecture
Detailed design
Another design cycle
Least abstract
Integration
Test
Moderately abstract
Specification
Software architecture
Module design
Another design cycle
Least abstract
Integration
Test
Moderately abstract

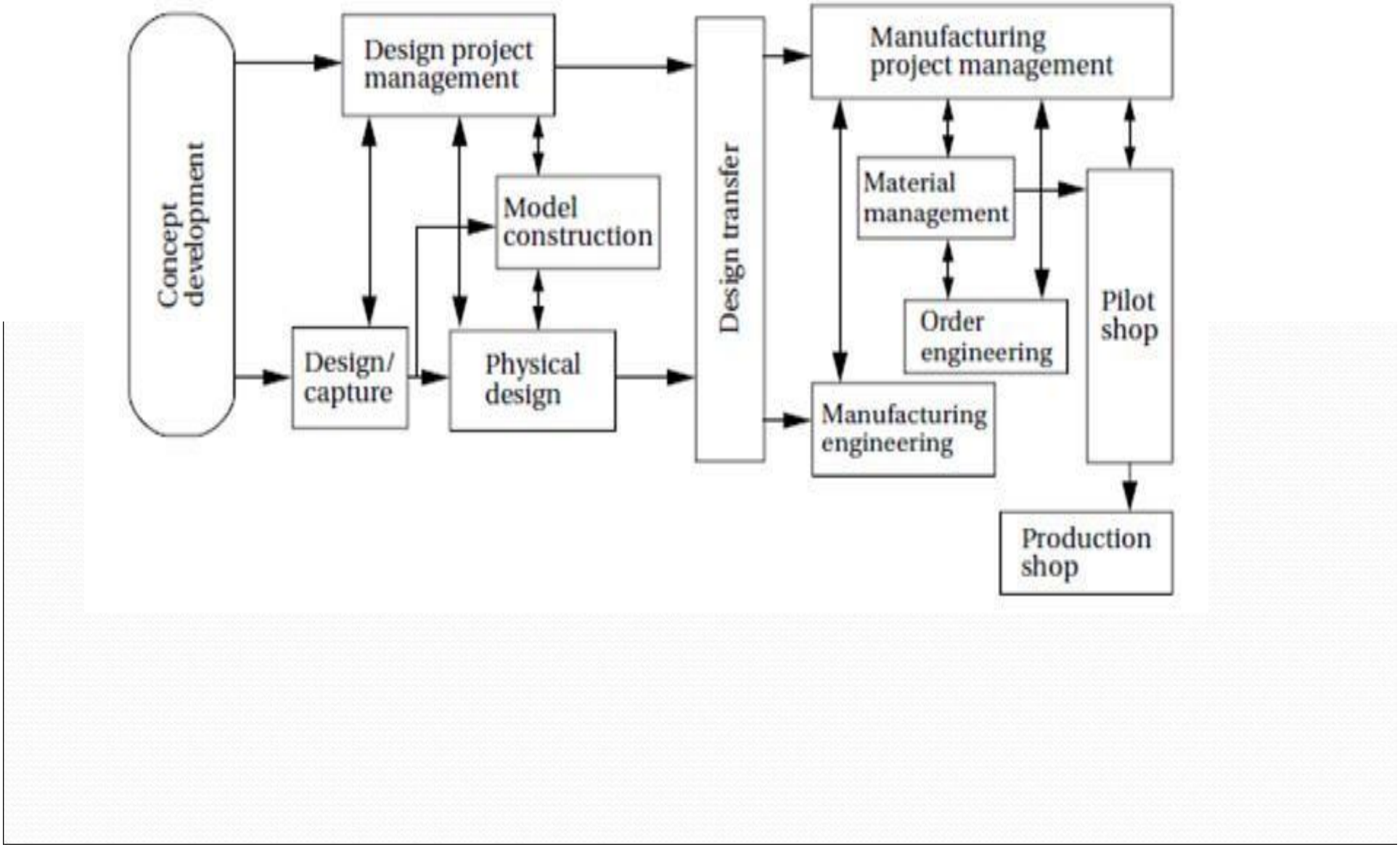
Concurrent engineering

- Reduced design time is an important goal for concurrent engineering.
- It eliminate "over-the-wall" design steps, one designer performs an isolated task and then throws the result to the next designer.

Concurrent engineering efforts are comprised of several elements.

- Cross-functional teams →include members from various disciplines (manufacturing, hardware and software design, marketing)
- Concurrent product→ realize the process activities .
- Designing various subsystems simultaneously, is reducing design time.
- Integrated project management→ensures that someone is responsible for the entire project.
- Early and continual supplier→ make the best use of suppliers' capabilities.
- Early and continual customer →ensure that the product meets customers' needs.

Concurrent Engineering Applied to Telephone Systems



1. **Benchmarking→** They compared themselves to competitors and found that it took them 30% longer to introduce a new product than their best competitors.
2. **Breakthrough improvement.**

- Increased partnership between design and manufacturing.
- Continued existence of the basic organization of design labs and manufacturing.
- Support of managers at least two levels above the working level.

3. **Characterization of the current process.**

- Too many design and manufacturing tasks were performed sequentially.

4. **Create the target process→** The core team created a model for the new development process.
5. **Verify the new process→** test the new process.
6. **Implement across the product line→**This activity required training of personnel, documentation of the new standards and procedures, and improvements to information systems.
7. **Measure results and improve→P**erformance of the new design was measured.

REQUIREMENTS ANALYSIS

- Requirements→It is a informal descriptions of what the customer wants.
- A functional requirement→ states what the system must do.
- A nonfunctional requirement→It can be physical size, cost, power consumption, design time, reliability, and so on.

Requirements of tests

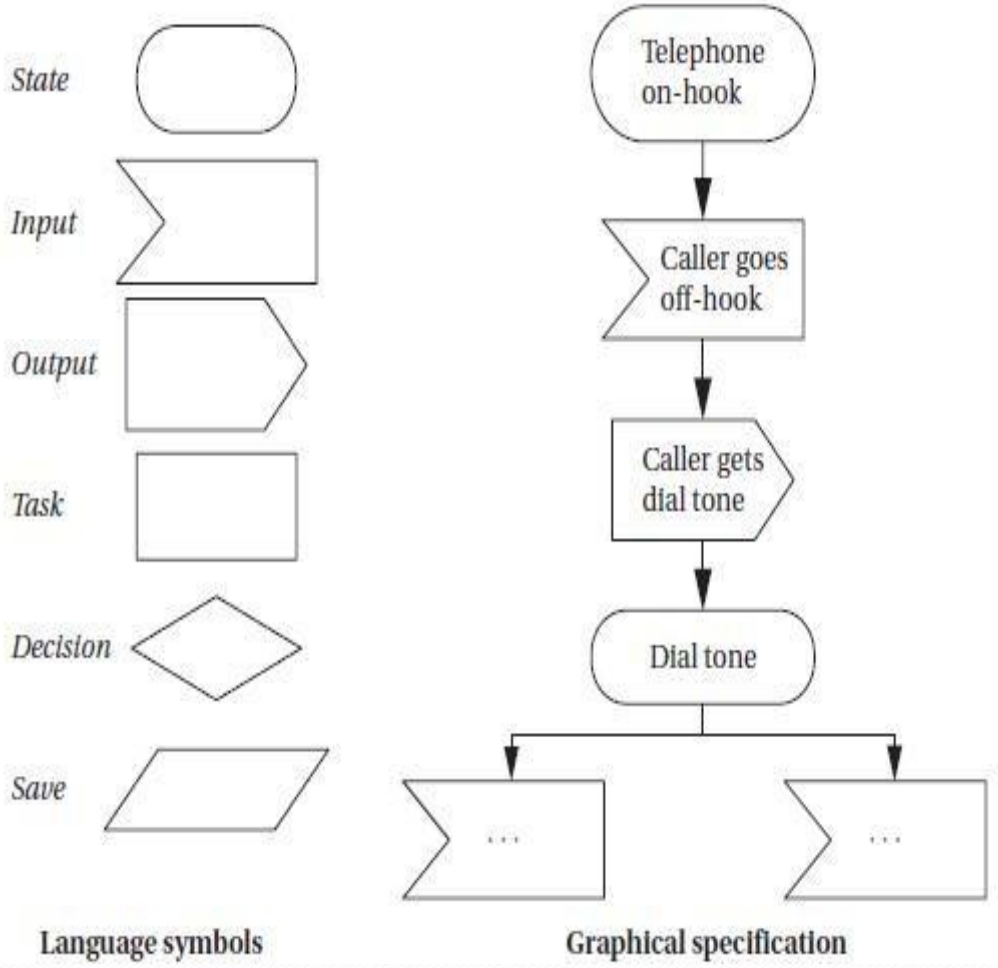
- Correctness→Requirements should not mistakenly describe what the customer wants.
- Unambiguousness→Requirements document should be clear and have only one plain language interpretation.
- Completeness→ Requirements all should be included.
- Verifiability→ cost-effective way to ensure that each requirement is satisfied in the final product.
- Consistency→One requirement should not contradict another requirement.
- Modifiability→The requirements document should be structured so that it can be modified to meet changing requirements without losing consistency.
- Traceability→Able to trace forward /backward from the requirements.

SPECIFICATIONS

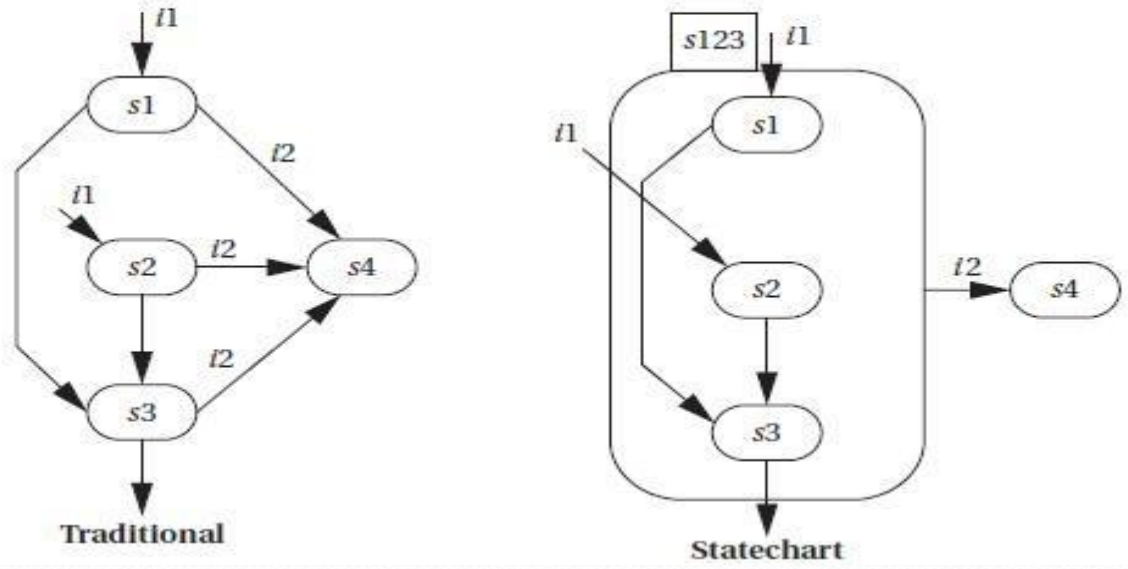
- Specifications→It is a detailed descriptions of the system that can be used to create the architecture.

Control-oriented specification languages

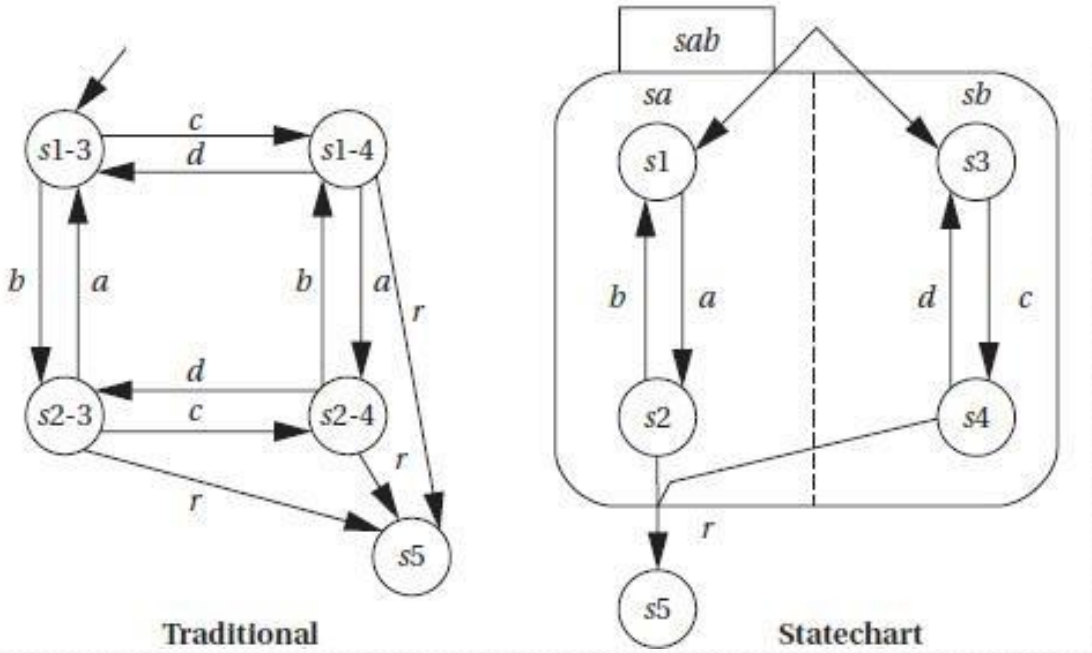
- SDL specifications include states, actions, and both conditional and unconditional transitions between states.
- SDL is an event-oriented state machine model.
- State chart has some important concepts.
- State charts allow states to be grouped together to show common functionality.



- ## Basic groupings(OR)
- State machine specifies that the machine goes to state s4 from any of s1, s2, or s3 when they receive the input i2.
- The State chart denotes this commonality by drawing an OR state around s1, s2, and s3 .
- Single transition out of the OR state s123 specifies that the machine goes to s4 when it receives the i2 input while in any state included in s123.
- Multiple ways to get into s123 (via s1 or s2), and transitions between states within the OR state (from s1 to s3 or s2 to s3).
- The OR state is simply a tool for specifying some of the transitions relating to these states.



- ## Basic groupings(AND)
- In the State chart, the AND state sab is decomposed into two components, sa and sb.
- When the machine enters the AND state, it simultaneously inhabits the state s1 of component sa and the state s3 of component sb.
- When it enters sab, the complete state of the machine requires examining both sa and sb.
- State s1-3 in the State chart machine having its sa component in s1 and its sb component in s3.
- When exit from cluster states go to s5 only when in the traditional specification, we are in state s2-4 and receive input r.



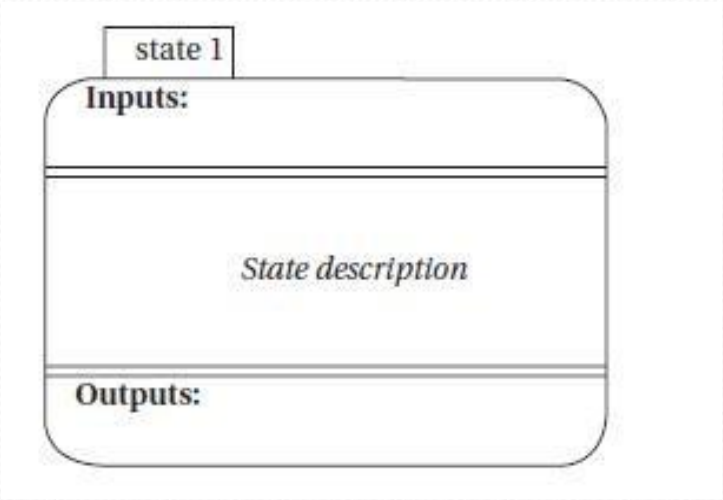
Advanced specifications

- It ensure the correctness and safety of this system.

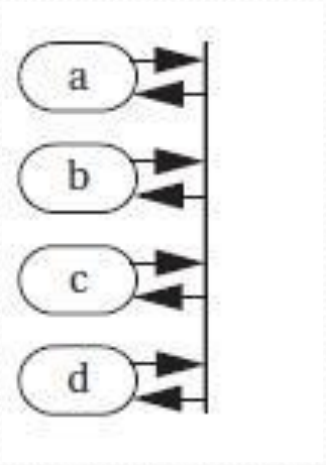
Ex→ Traffic Alert and Collision Avoidance System(TCAS)

- It is a collision avoidance system for aircraft.
- TCAS unit in an aircraft keeps track of the position of other nearby aircraft.
- It uses pre-recorded voice ("DESCEND!) commands for mid-air collision.
- TCAS makes sophisticated decisions in real time and is clearly safety critical.
- It must detect as many potential collision events as possible .
- It must generate a few false alarms ,at extreme maneuvers in potentially dangerous.

TCAS-II specification(RSML Language)

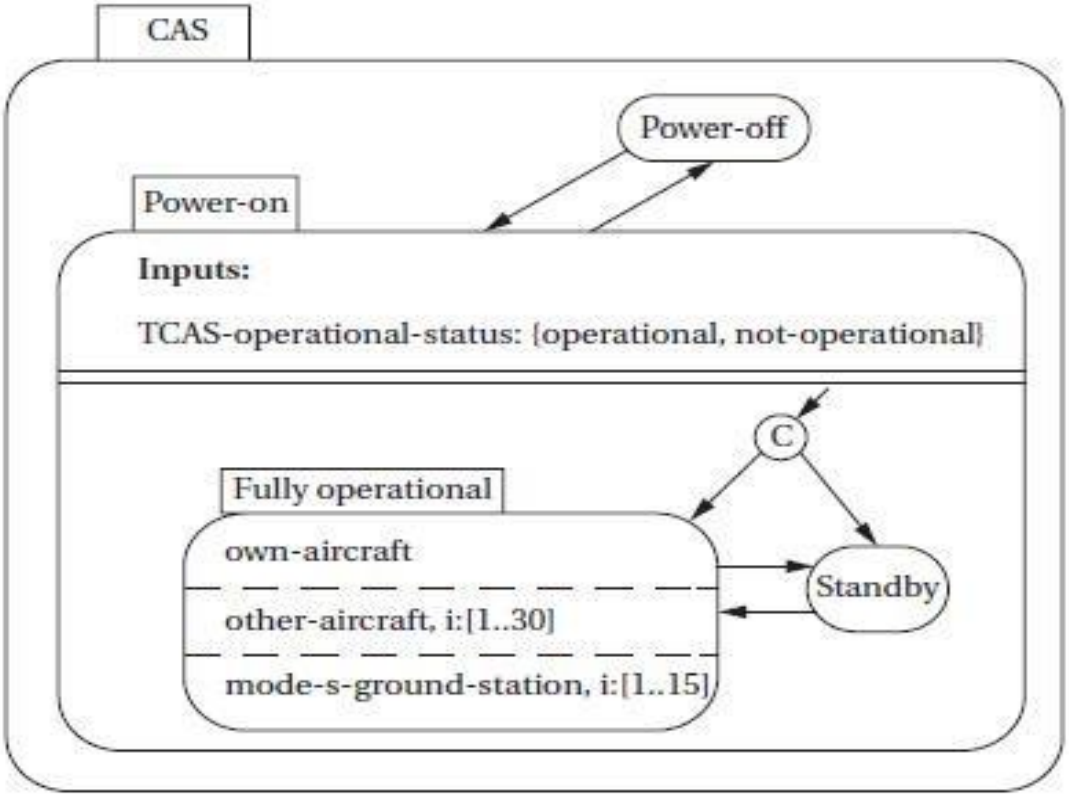


Transition states



Collision Avoidance system

- The system has Power-off and Power-on states .
- In the power on state, the system may be in Standby or Fully operational mode.
- In the Fully operational mode, three components are operating in parallel, as specified by the AND state.
- The own aircraft subsystem to keep track of up to 30 other aircraft.
- Subsystem to keep track of up to 15 Mode S ground stations, which provide radar information.



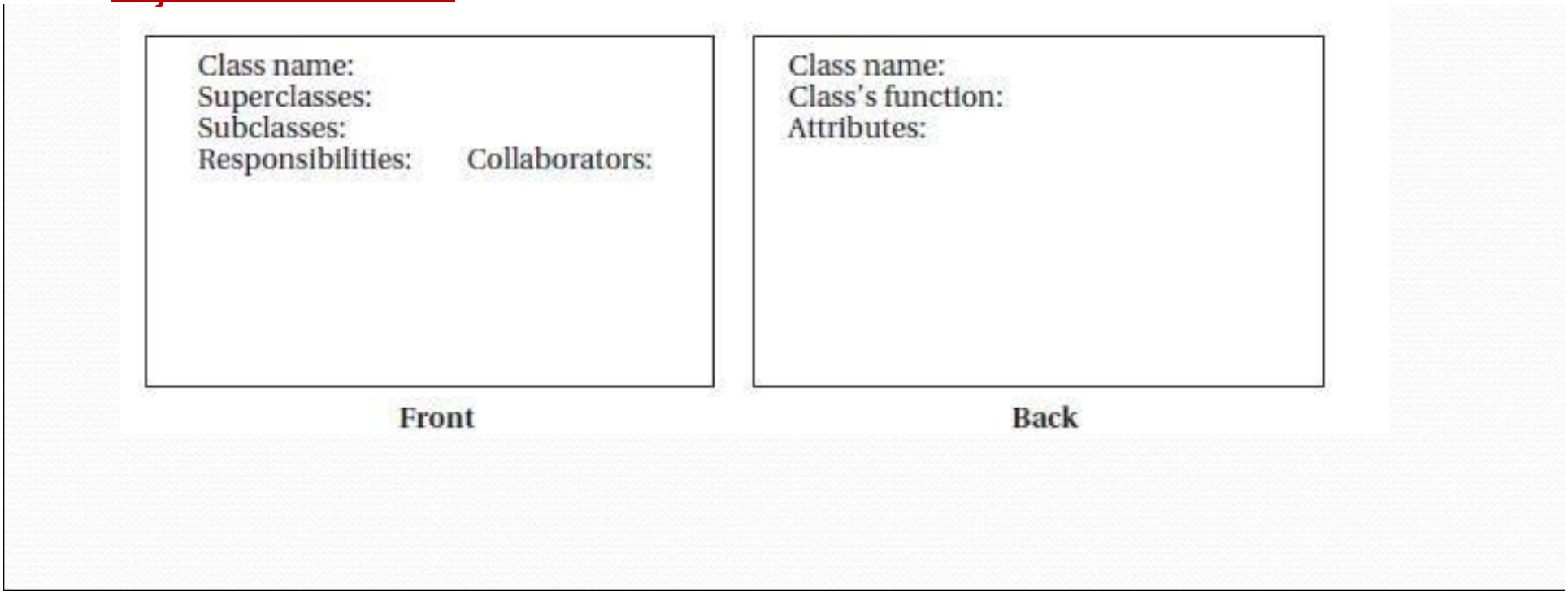
SYSTEM ANALYSIS AND ARCHITECTURE DESIGN

- The CRC card methodology analyze and understanding the overall structure of a complex system.

CRC cards

- Classes define the logical groupings of data and functionality.
- Responsibilities describe what the classes do.
- Collaborators are the other classes with which a given class works.
- It has space to write down the class name, its responsibilities and collaborators, and other information.

Layout of CRC card



- A class may represent a real-world object of the system.
- A class has both an internal state and a functional interface.
- The functional interface describes the class's capabilities.
- The responsibility set is describing that functional interface.
- The collaborators of a class are simply the classes that it talks or calls upon to help it do its work.
- CRC card Analysis Process

1. Develop an initial list of classes→Write down the class name and functions of it.
2. Write an initial list of responsibilities and collaborators.
3. Create some usage scenarios→describe what the system does.
4. Walk through the scenarios→Each person on the team represents one or more classes.
5. Refine the classes, responsibilities, and collaborators→ make changes to the CRC cards.
6. Add class relationships→ subclass and super-class can be added to the cards.

Ex:Elevator system

1. One passenger requests a car on a floor, gets in the car when it arrives, requests another floor, and gets out when the car reaches that floor.
2. One passenger requests a car on a floor, gets in the car when it arrives, and requests the floor that the car is currently on.
3. A second passenger requests a car while another passenger is riding in the elevator.
4. Two people push floor buttons on different floors at the same time.
5. Two people push car control buttons in different cars at the same time.

Class	*Responsibilities*	*Collaborators*
Elevator car*	Moves up and down	Car control, car sensor, car control sender
Passenger*	Pushes floor control and car control buttons	Floor control, car control
Floor control*	Transmits floor requests	Passenger, floor control reader
Car control*	Transmits car requests	Passenger, car control reader
Car sensor*	Senses car position	Scheduler
Car state	Records current position of car	Scheduler, car sensor
Floor control reader	Interface between floor control and rest of system	Floor control, scheduler
Car control reader	Interface between car control and rest of system	Car control, scheduler
Car control sender	Interface between scheduler and car	Scheduler, elevator car
Scheduler	Sends commands to cars based upon requests	Floor control reader, car control reader, car control sender, car state

Capability Maturity Model (CMM)

- It is used to measuring the quality of an organization's software development.

1. Initial→A poorly organized process, with very few well-defined processes. Success of a project depends on the efforts of individuals, not the organization itself.
2. Repeatable→ provides basic tracking mechanisms to understand cost, scheduling .
3. Defined→The management and engineering processes are documented and standardized.
4. Managed→detailed measurements of the development process and product quality.
5. Optimizing→feedback from detailed measurements is used to continually improve the organization's processes.

- Verifying the specification→Discovering bugs early is crucial because it prevents bugs from being released to customers, minimizes design costs, and reduces design time.
- Validation of specifications→creating the requirements, including correctness, completeness, consistency, and so on

Design reviews

- The review leader coordinates the pre-meeting activities, the design review itself, and the post-meeting follow-up.
- The reviewer records the minutes of the meeting so that designers and others know which problems need to be fixed.
- The review audience studies the component.

DESIGNING WITH COMPUTING PLATFORM

System Architecture

- The architecture of an embedded computing system includes both hardware and software elements

HARDWARE

- ***CPU*** →The choice of the CPU is one of the most important, but it can be considered the software that will execute on the machine.
- ***Bus*** →The choice of a bus is closely tied to that of a CPU,bus can handle the traffic.
- ***Memory***→Selection depends total size and speed of the memory will play a large part in determining system performance.
- **Input and output devices**→ Dependig upon the system requirements

SOFTWARE

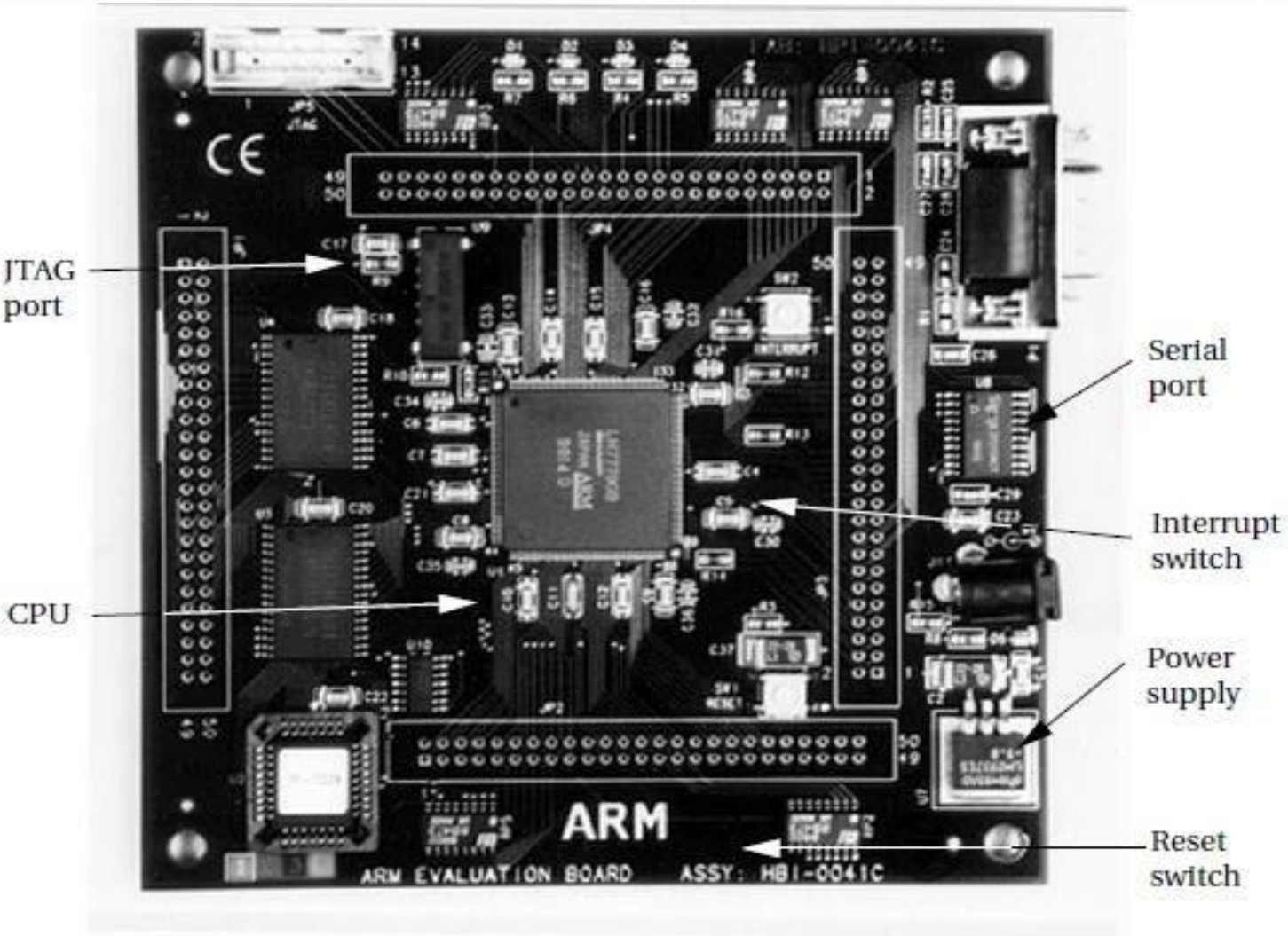
Run Time components

- It is a critical part of the platform.
- An operating system is required to control CPU and its multiple processes .
- A file system is used in many embedded systems to organize internal data and interface with other systems

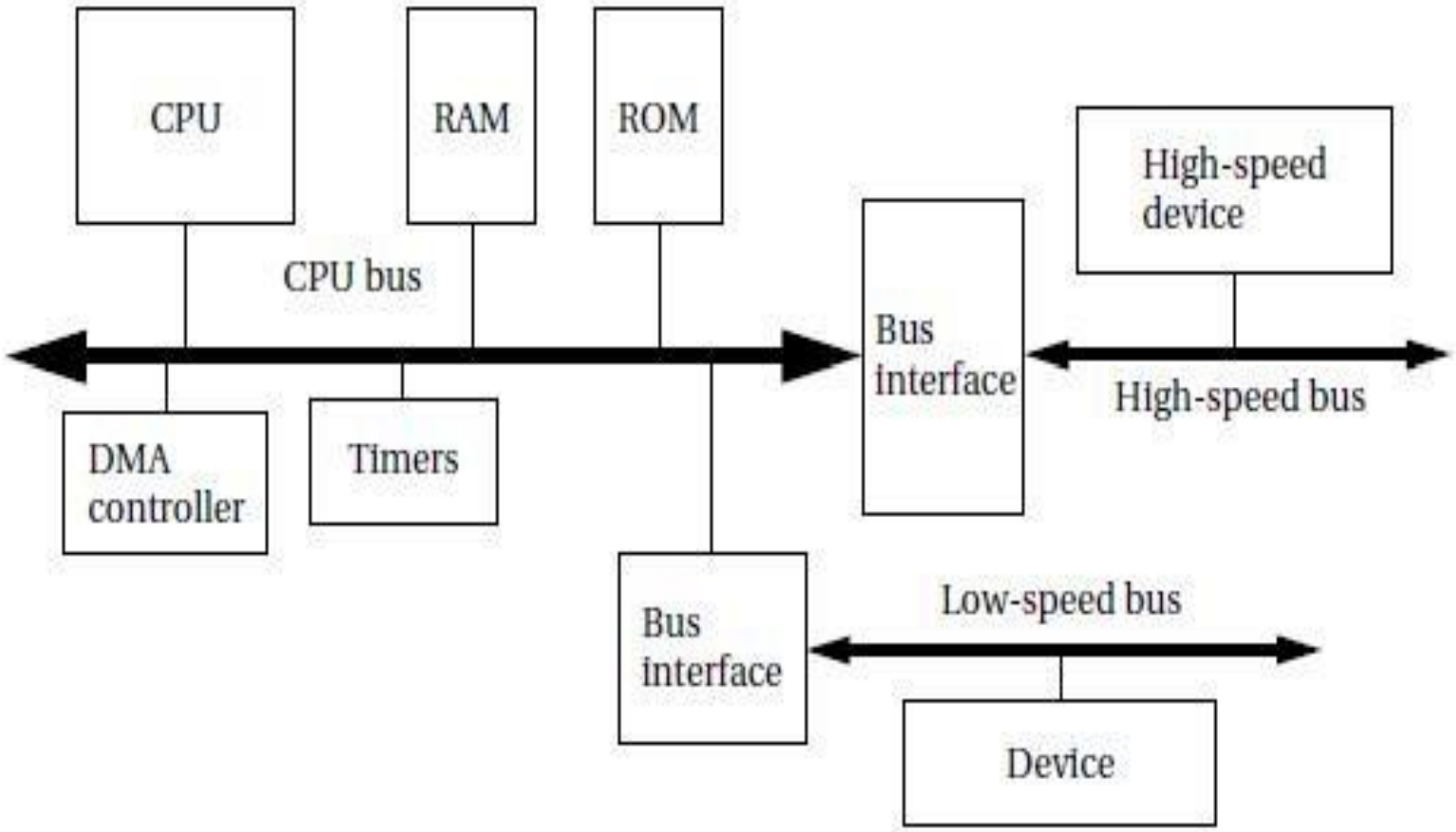
Support components

- It is a complex hardware platform.
- Without proper code development and operating system, the hardware itself is useless.

ARM evaluation board



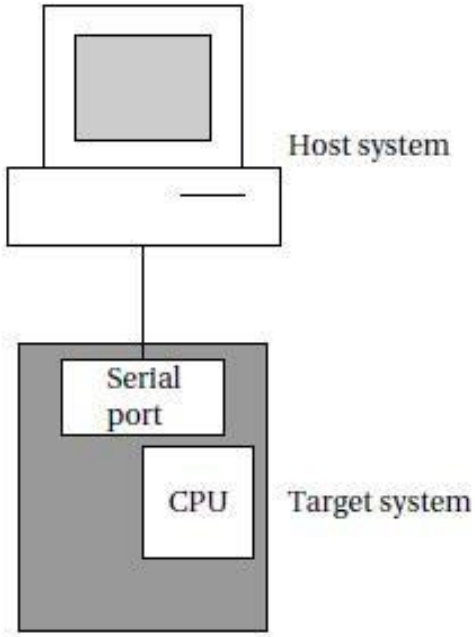
1.10.2)The PC as a Platform



- CPU → provides basic computational facilities.
- RAM → is used for program storage.
- ROM → holds the boot program.
- DMA → controller provides DMA capabilities.
- Timers → used by the operating system for a variety of purposes.
- High-speed bus → connected to the CPU bus through a bridge, allows fast devices to communicate with the rest of the system.
- low-speed bus → provides an inexpensive way to connect simpler devices.

Development Environments

- Development process → used to make a complete design of the system.
- It guides the developers how to design a system .
- An embedded computing system has CPU ,memory, I/O devices.
- Development of embedded system have both hardware& software.
- The software development on a PC or workstation known as a host.



•The host and target are frequently connected by a USB link.
•The target must include a small amount of software to talk to the host system.

Functions of Host system

- Load programs into the target
- Start and stop program execution on the target
- Examine memory and CPU registers.

Cross-Compiler

- Compiler→ kind of software that translate one form of pgm to another form of pgm.
- Cross Compiler→is a compiler that runs on one type of machine but generates
- code for another
- After compilation, the executable code is downloaded to the embedded system by a serial link.
- A PC or workstation offers a programming environment .
- But one problem with this approach emerges when debugging code talks to I/O devices.
- Testbench program→ can be built to help debug the embedded code.
- It may also take the output values and compare them against expected values.

Debugging Techniques

- It is the process of checking errors and correcting those errors.
- It can be done by compiling and executing the code on a PC or workstation.
- It can be performed by both H/W and S/W sides.

Software debugging tools

1. Serial Port tool

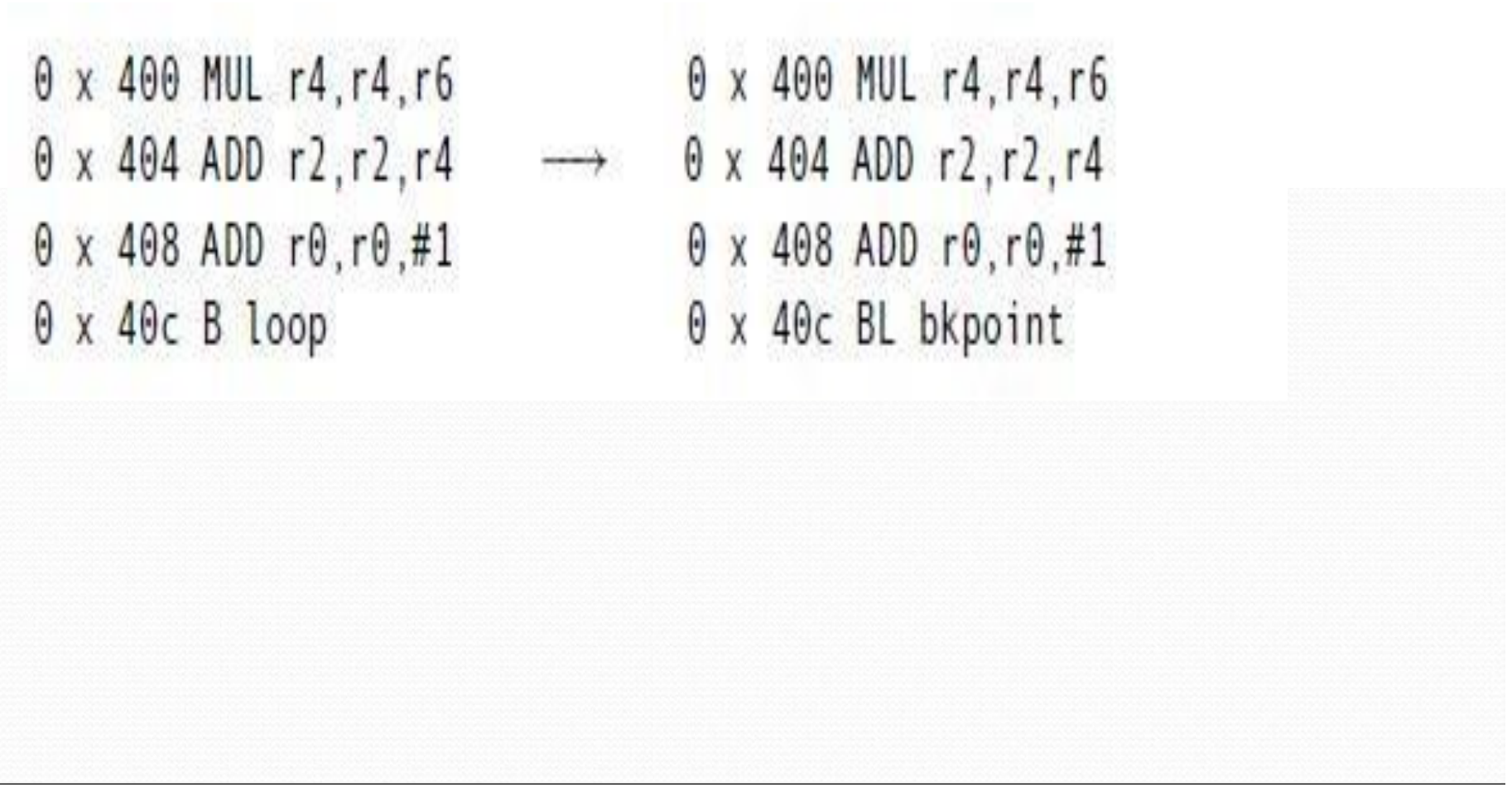
- It will perform the debugging process from the initial state of embedded system design.
- It can be used not only for development debugging but also for diagnosing problems in the field.

2. Breakpoints tool

- user to specify an address at which the program's execution is to break.
- When the PC reaches that address, control is returned to the monitor program.
- From the monitor program, the user can examine and/or modify CPU registers, after which execution can be continued.

▪Breakpoint is a location in memory at which a program stops executing and returns to the debugging tool or monitor program.

To establish a breakpoint at location 0x40c in some ARM code, replaced the branch (B) instruction with a subroutine call (BL) to the breakpoint handling routine



- Hardware debugging tools
- Hardware can be deployed to give a clearer view on what is happening when the system is running.

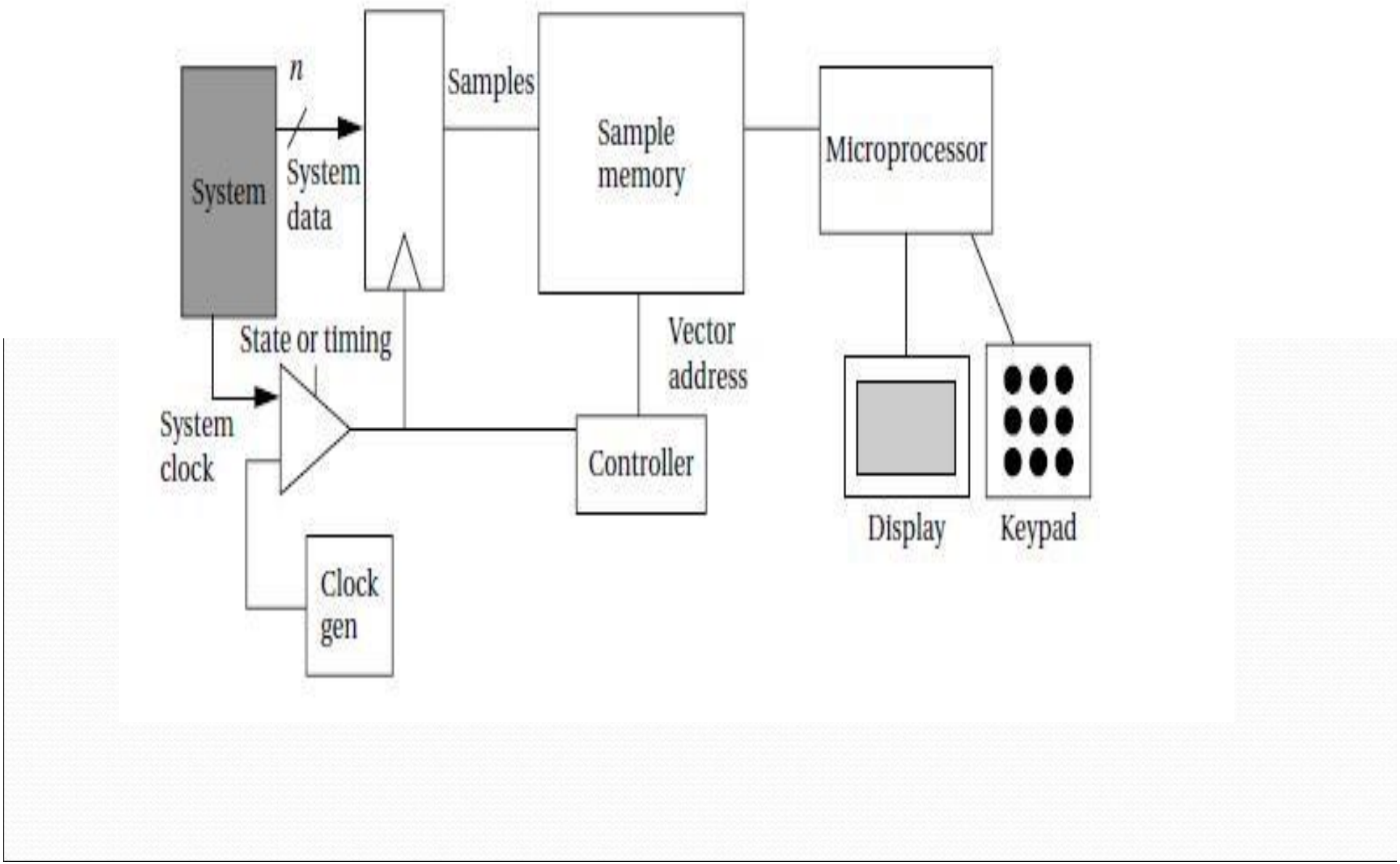
1. Microprocessor In-Circuit Emulator (ICE)

- It is a specialized hardware tool that can help debug software in a working embedded system.
- In-circuit emulator is a special version of the microprocessor that allows its internal registers to be read out when it is stopped

2. Logic Analyer

- The analyzer can sample many different signals simultaneously but can display only 0, 1, or changing values for each.
- The logic analyzer records the values on the signals into an internal memory and then displays the results on a display once the memory is full.

Architecture of a logic analyzer



Data modes of logic analyzer

State modes

- State mode represent different ways of sampling the values.
- It uses the own clock to control sampling
- It samples each signal only one per clock cycle.
- It has less memory to store a given number of system clock.

Timing modes

- Timing mode uses an internal clock that is fast enough to take several samples per clock period in a typical system.

1.10.5)Debugging Challenges

- Logical errors in software can be hard to track down and it will create many problems in real time code.
- Real-time programs are required to finish their work within a certain amount of time.
- Run time pgm run too long, they can create very unexpected behavior.
- Missing of Deadline makes debugging process as difficult.

Consumer Electronic Architecture

- Consumer electronic refers to any device containing an electronic circuit board that is intended for eneryday use by individuals.
- Eg→TV,cameras,digital cameras,calculators,DVDs,audio devices,smart phones etc..,

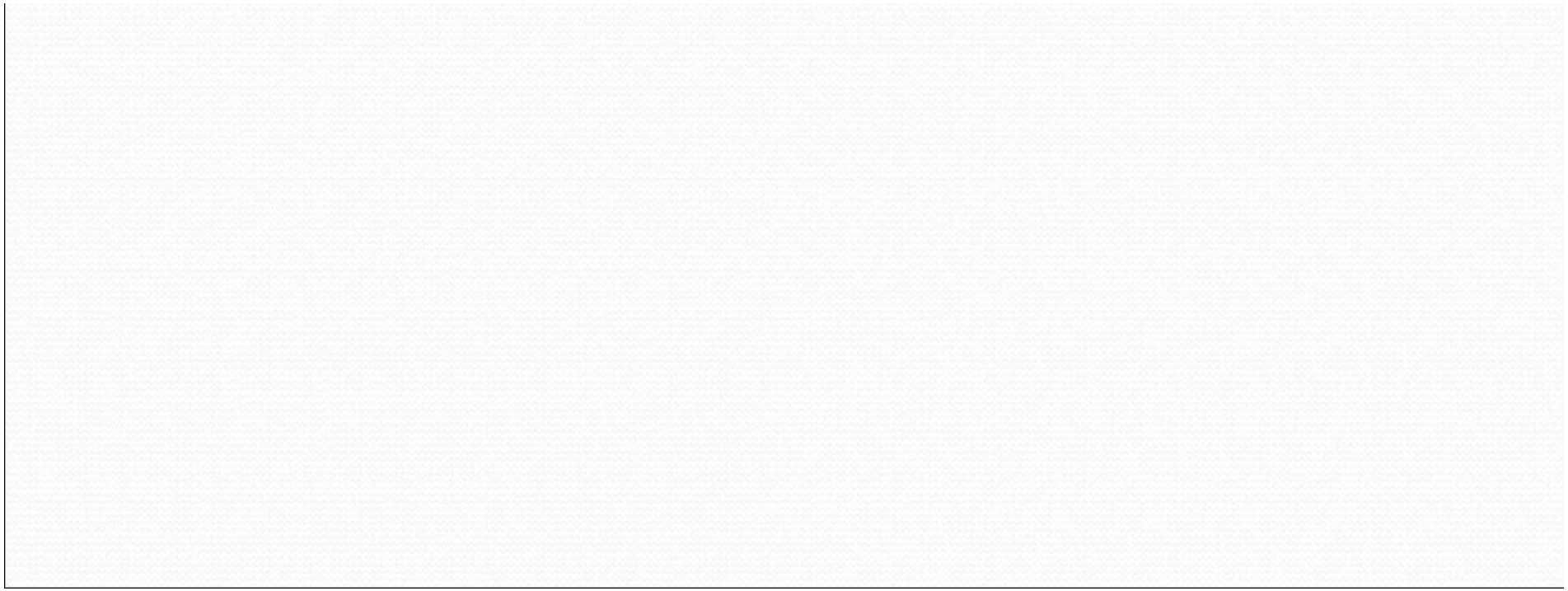
1.11..1)Functional Requirements

1. *Multimedia*
- The media may be audio, still images, or video.
- These multimedia objects are generally stored in compressed form and must be uncompressed to be played .
- Eg→ multimedia compression standards (MP3,Dolby Digital(TM))
- audio; JPEG for still images; MPEG-2, MPEG-4, H.264, etc. for video.

2. *Data storage and management*→ People want to select what multimedia objects they save or play, data storage goes hand-in-hand with multimedia capture and display. Many devices provide PC-compatible file systems so that data can be shared more easily.
3. *Communications*→ Communications may be relatively simple, such as a USB
- and another is Ethernet port or a cellular telephone link.

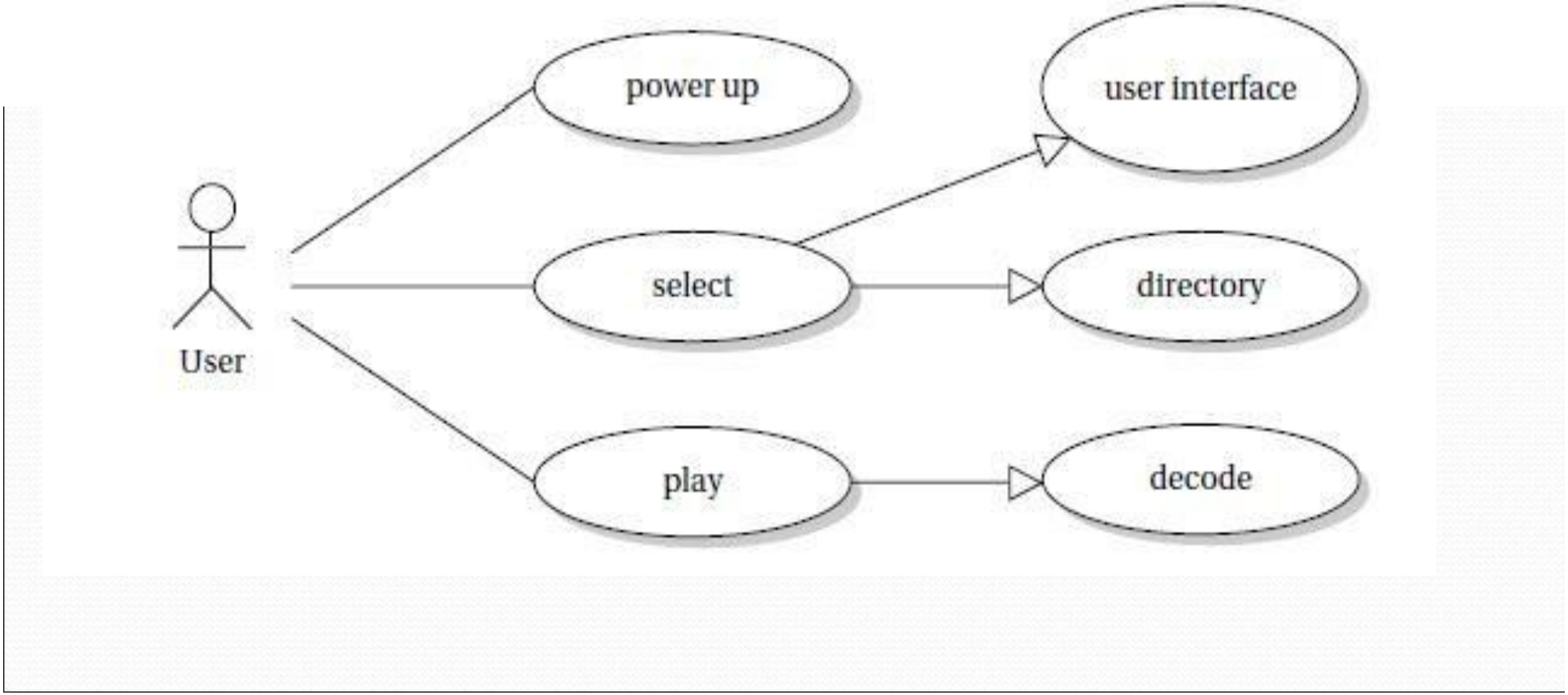
Non-Functional Requirements

- Many devices are battery-operated, which means that they must operate under strict energy budgets.
- Battery(75mW) →support not only the processors but also the display, radio, etc.
- Consumer electronics must also be very inexpensive but provide very high performance.



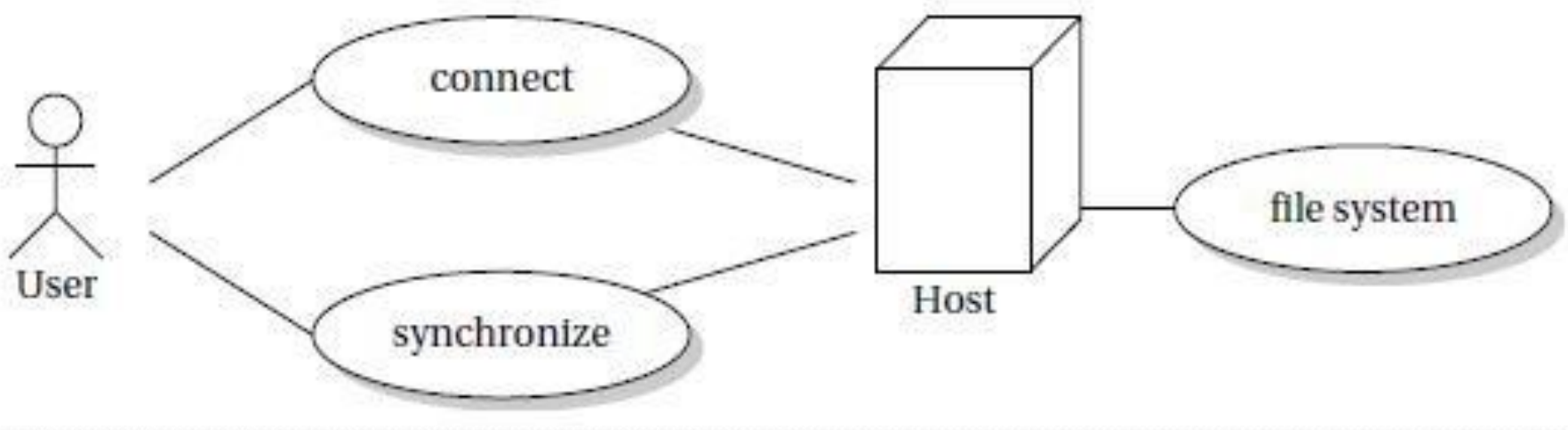
Use case for playing multimedia

- use case for selecting and playing a multimedia object (audio clip, a picture,etc.).
- Selecting an object makes use of both the user interface and the file system.
- Playing also makes use of the file system as well as the decoding subsystem and I/O subsystem.

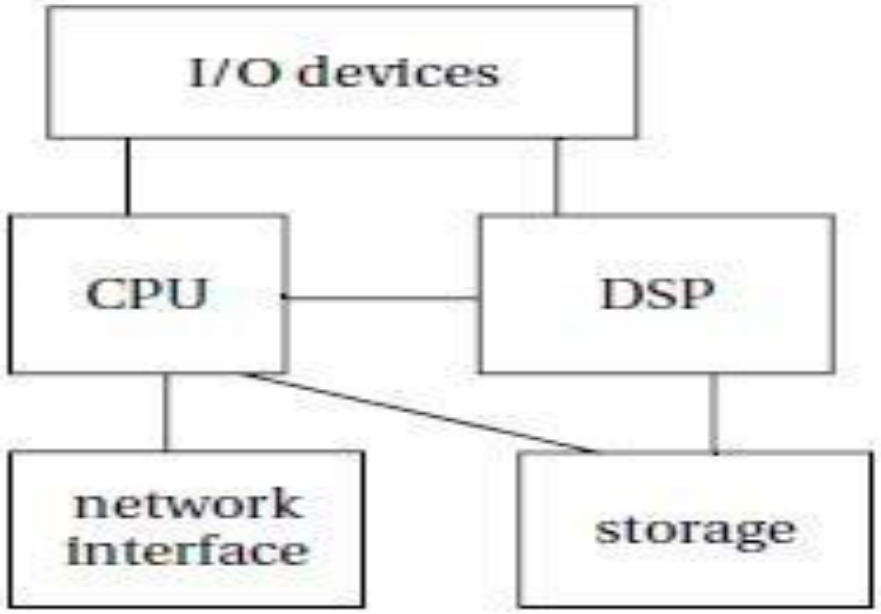


Use case of synchronizing with a host system

- use case for connecting to a client.
- The connection may be either over a local connection like USB or over the Internet.
- Some operations may be performed locally on the client device
- most of the work is done on the host system while the connection is established



Functional architecture of Consumer Electronics Device(CED)



- It is a two-processor architecture.
- If more computation is required, more DSPs and CPUs may be added.
- The RISC-CPU runs the operating system, runs the user interface, maintains the file system, etc.
- DSP→ it is a programmable one, which performs signal processing.
- Operating system→ runs on the CPU must maintain processes and the file system.
- Depending on the complexity of the device, the operating system may not need to create tasks dynamically.
- If all tasks can be created using initialization code, the operating system can be made smaller and simpler.

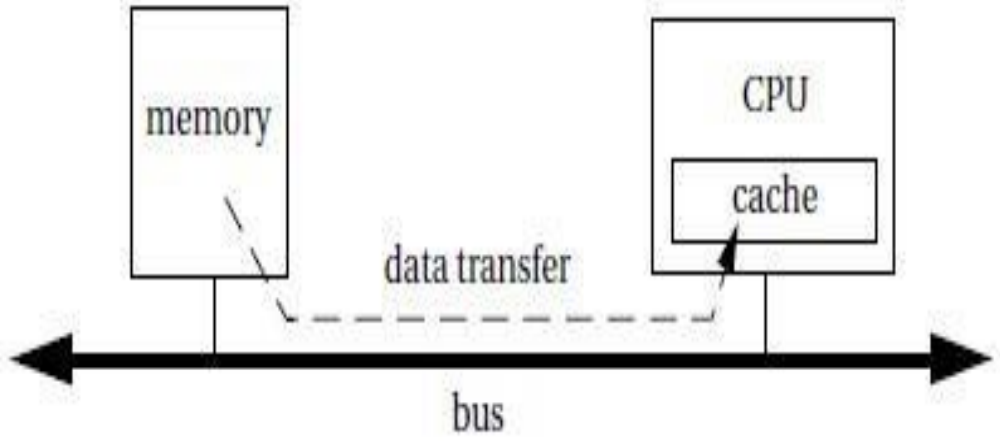
1.11.4 Flash File Systems

- Many consumer electronics devices use flash memory for mass storage.
- Flash memory is a type of semiconductor memory ,unlike DRAM or SRAM, provides permanent storage.
- Values are stored in the flash memory cell as electric charge using a specialized capacitor that can store the charge for years.
- The file system of a device is typically shared with a PC.
- Standard file system→has two layers.bottom layer handles physical reads and writes on the storage device and the top layer provides a logical view of the file system.
- Flash file system→imposes an intermediate layer that allows the logical-to-physical mapping of files to be changed.

Platform-Level Performance Analysis

- System-Level Performance involves much more than the CPU.
- To move data from memory to the CPU to process it. To get the data from memory to the CPU we must.

1. Read from the memory.
2. Transfer over the bus to the cache.
3. Transfer from the cache to the CPU.



- The performance of the system based on Bandwidth of the system.
- We can increase bandwidth in two ways:

1) By increasing the clock rate of the bus
2) By increasing the amount of data transferred per clock cycle.

For example, bus to carry four bytes or 32 bits per transfer, we would reduce the transfer time to 0.058 s. If we also increase the bus clock rate to 2 MHz, then we would reduce the transfer time to 0.029 s ,which is within our time budget for the transfer.

$$t=TP$$

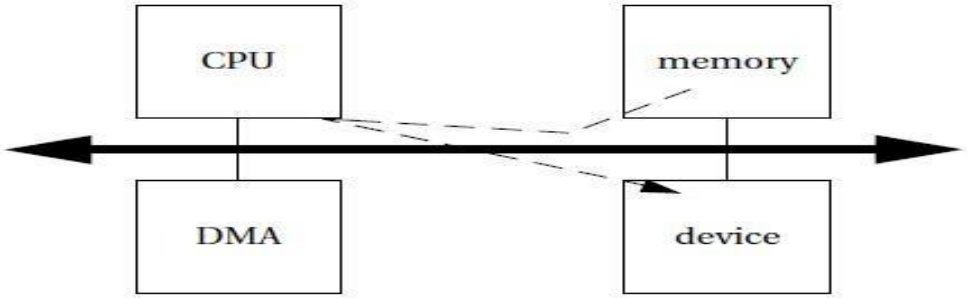
t→bus cycle counts

T→bus cycles.

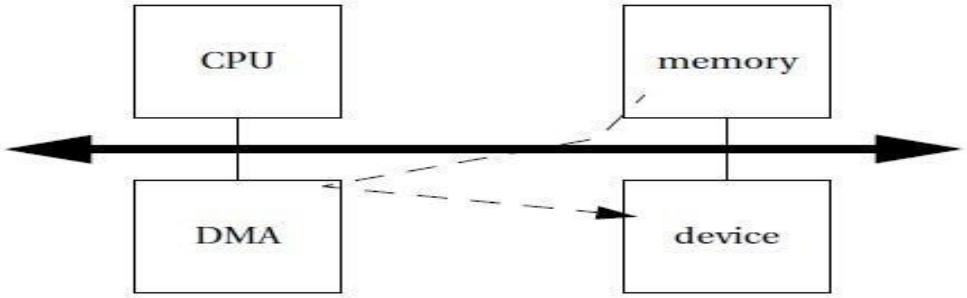
p→bus clock period

Parallelism

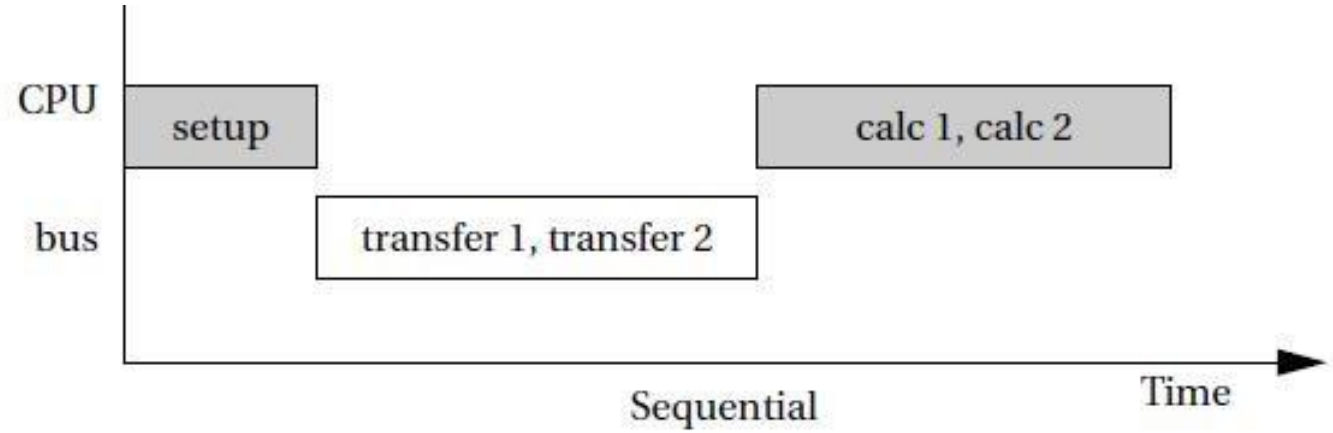
- Direct memory access is a example of parallelism.
- DMA was designed to off-load memory transfers from the CPU.
- The CPU can do other useful work while the DMA transfer is running.

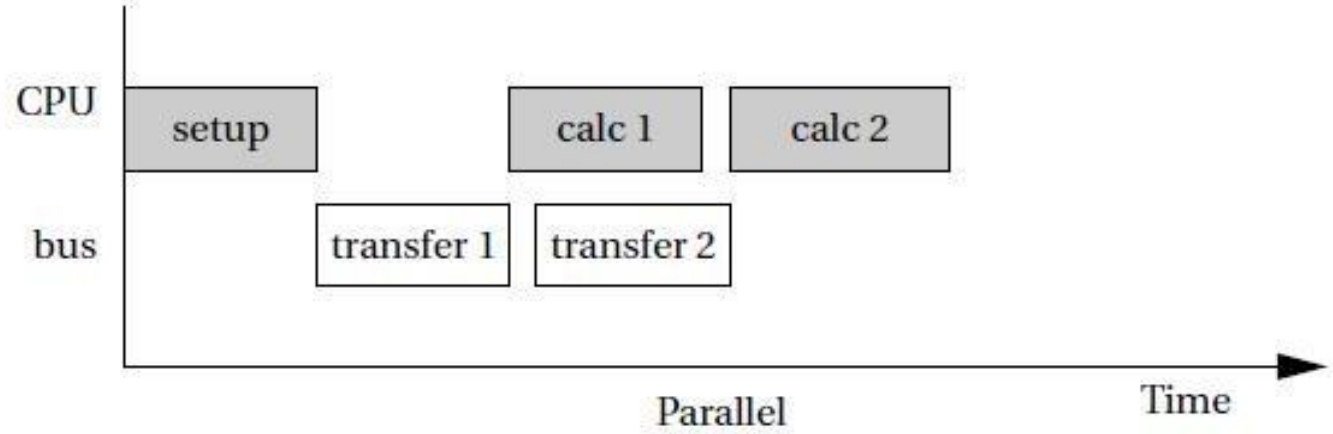


transfer without DMA



transfer without DMA


CPU
setup
calc 1, calc 2
bus
transfer 1, transfer 2
Time
Sequential


CPU
setup
calc 1
calc 2
bus
transfer 1
transfer 2
Time
Parallel

UNIT II

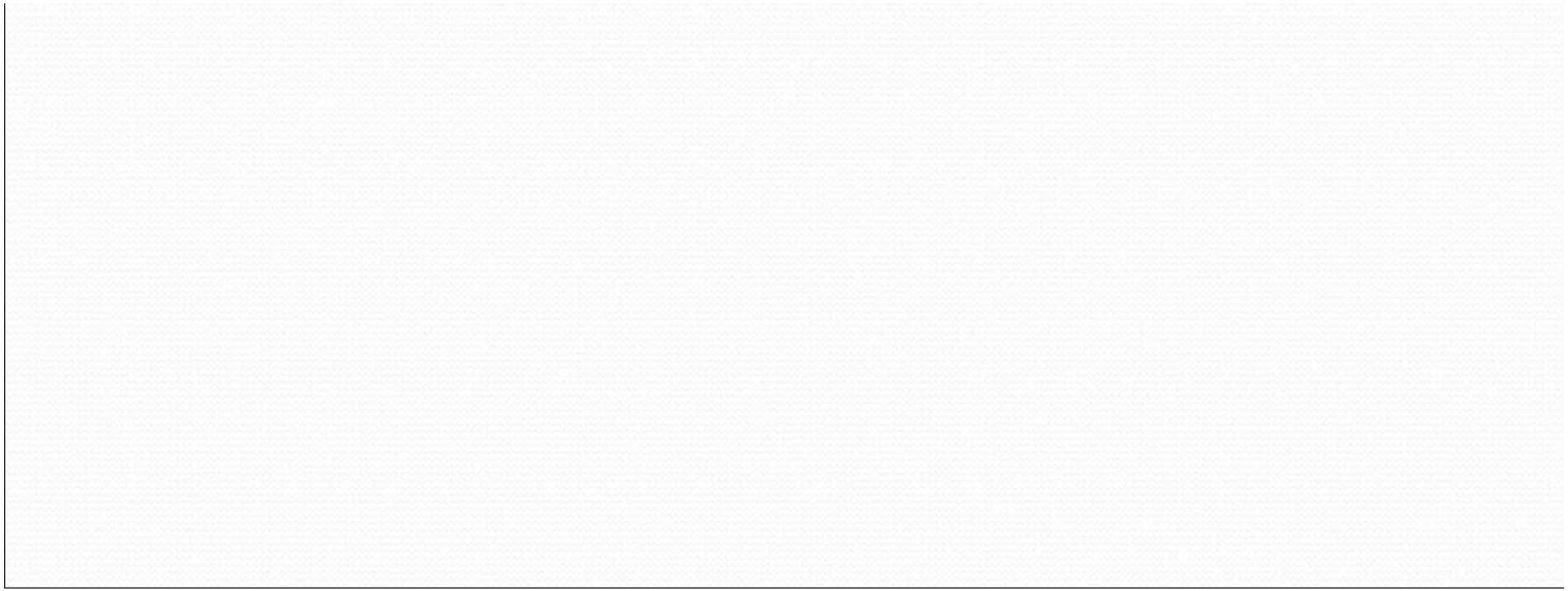
ARM PROCESSOR AND PERIPHERALS

ARM Architecture Versions – ARM Architecture – Instruction Set – Stacks and Subroutines – Features of the LPC 214X Family – Peripherals – The Timer Unit – Pulse Width Modulation Unit – UART – Block Diagram of ARM9 and ARM Cortex M3 MCU.

ARM Architecture Versions

- The ARM processor is a Reduced Instruction Set Computer (RISC).
- The first ARM processor was developed at Acorn Computers Limited, of Cambridge, England, between October 1983 and April 1985. It is very simple architecture.
- At that time, and until the formation of Advanced RISC Machines Limited (which later was renamed simply ARM Limited) in 1990, ARM stood for Acorn RISC Machine

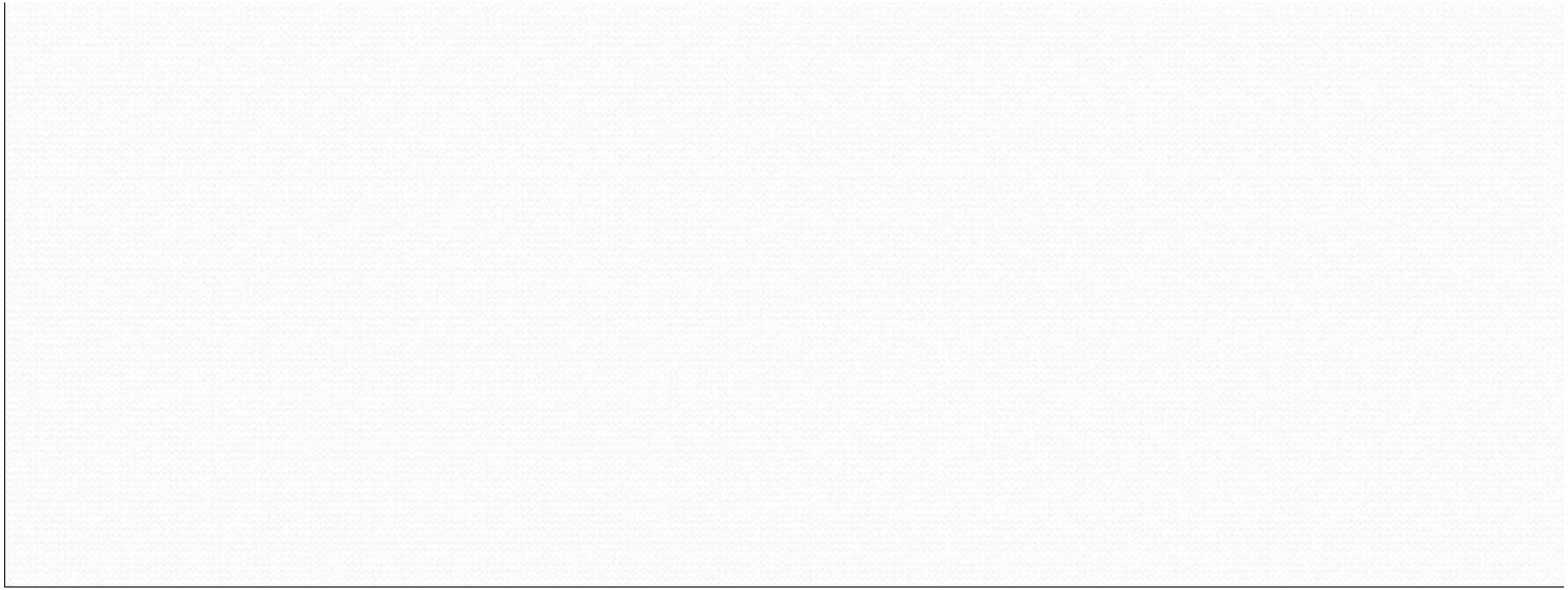
- Second, both ARM ISA and pipeline design are aimed to minimize the energy consumption.
- Third, the ARM architecture is highly modular only mandatory component of ARM processor is the integer pipeline, others are optional. This gives more flexibility in application dependent architecture



Revision	Example core implementation	ISA Enhancement
ARM v1	ARM1	•First ARM Processor •26bit addressing
ARMv2	ARM2	•32bit multiplier •32bit coprocessor support
ARMv2a	ARM3	•On chip cache •Atomic swap instruction •Coprocessor 15 for cache management
ARMv3	ARM6 and ARM7DI	•32 bit addressing •Separate cpsr (current Program status register)and spsr (Saved program status register) •New modes undefined instruction and abort •MMU support(Memory Management Unit)

ARMv3M	ARM7M	Signed and un signed long multiply instruction
ARMv4	Strong ARM	•load store instructions for signed half words/bytes •Reserve SWI(software interrupt) space fro architecturally define operations. •26 bit addressing mode no longer supported
ARMv4T	ARM7TDMI and ARM9T	•Thumb
ARMV5TE	ARM9E AND ARM10E	•Superset of ARM •Enhanced multiply instructions •Extra DSP type instruction •Faster multiply instruction

ARM V5tej	ARM7EJ & ARM926EJ	Java Acceleration
ARMv6	ARM11	•Improved Multiprocessor instructions •Unaligned and Mixed endian data handling



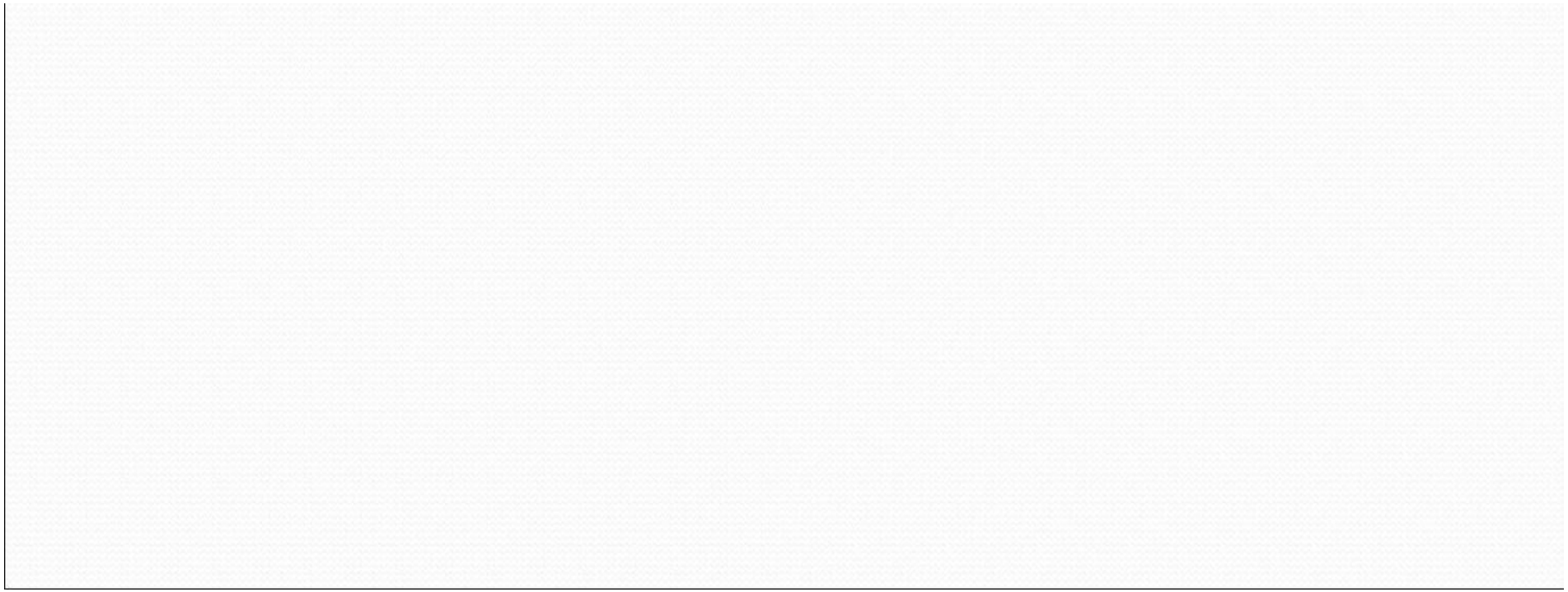
ARM processor Features

Terms	Extention
X	Family or series
Y	Memory Management
Z	Cache
T	16bit thumb decoder
D	Jtag Debugger
M	Fast multiplier
I	Embedded In circuit Emulator
E	Enhanced Instruction for DSP
J	Jazelle
F	Vector floating point unit
S	Synthesizable version

ARM 7family

- ARM7 core has a von neumann style architecture
- ARM7 TDMI is first processor introduced in 1995 by ARM
- It provide a very good performance to power ratio
- ARM7TDMI-S has the synthesizable
- ARM720T is the most fexible member of ARM7 family because it include MMU. MMU handle both platforms Linux and windows
- It having unified 8k cache and vector table are relocated depend on the priority

- ARM7EJS processor, also synthesizable. Its having five stage pipeline and execute ARMv5TEJ instruction
- This version only support java acceleration.



ARM9 family

- The ARM9 family was announced in 1997
- ARM9 has five stage pipeline and high clock frequencies
- Memory have been redesign Harvard architecture
- ARM9 process includes cache and MMU
- Operating system requiring virtual memory support
- ETM (Embedded Trace Macrocell) which allows a developer to trace instruction and data execution in real time operation. So that debugging is done during the critical time segments.
-

- ARM946E-S include TCM, cache and MPU. The size of the TCM and cache are configurable
- The processor is designed for the embedded control application that require deterministic real time response
- ARM926EJ-S synthesizable processor core, announced in 2000
- It a java enable device such as 3G phones and personal digital assistant

RM10 FAMILY

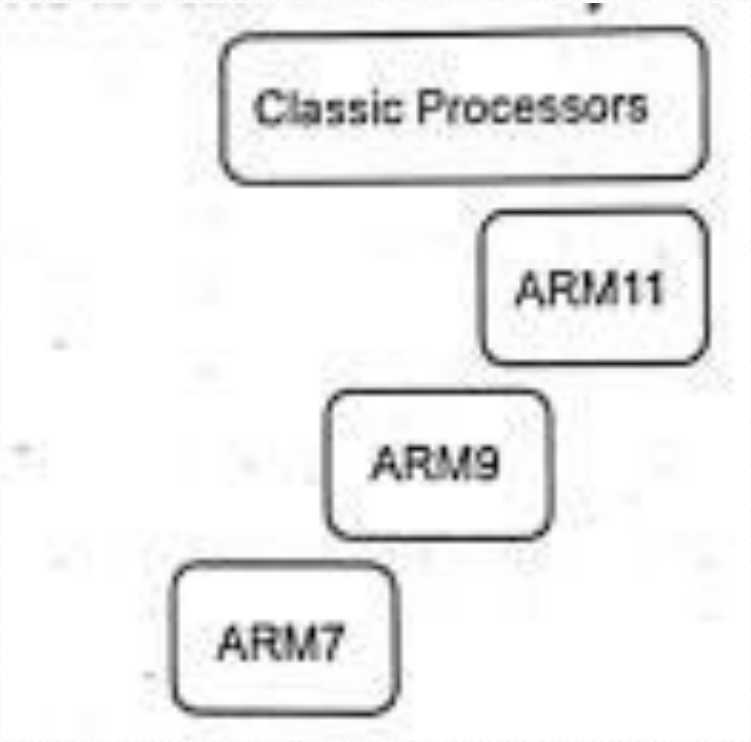
- The ARM10 announced in 1997 was designed for performance
- It extended version of 6 stage pipeline
- Vector floating point unit which adds a seventh stage to the ARM10 pipeline
- VFP combined with IEEE 754.1985 floating point
- ARM1020 E it includes E instruction. it having cache, VFP and MMU
- ARM1026EJ-S is similar to ARM926EJ-S . But ARM10 is flexible when compare to ARM9

ARM11

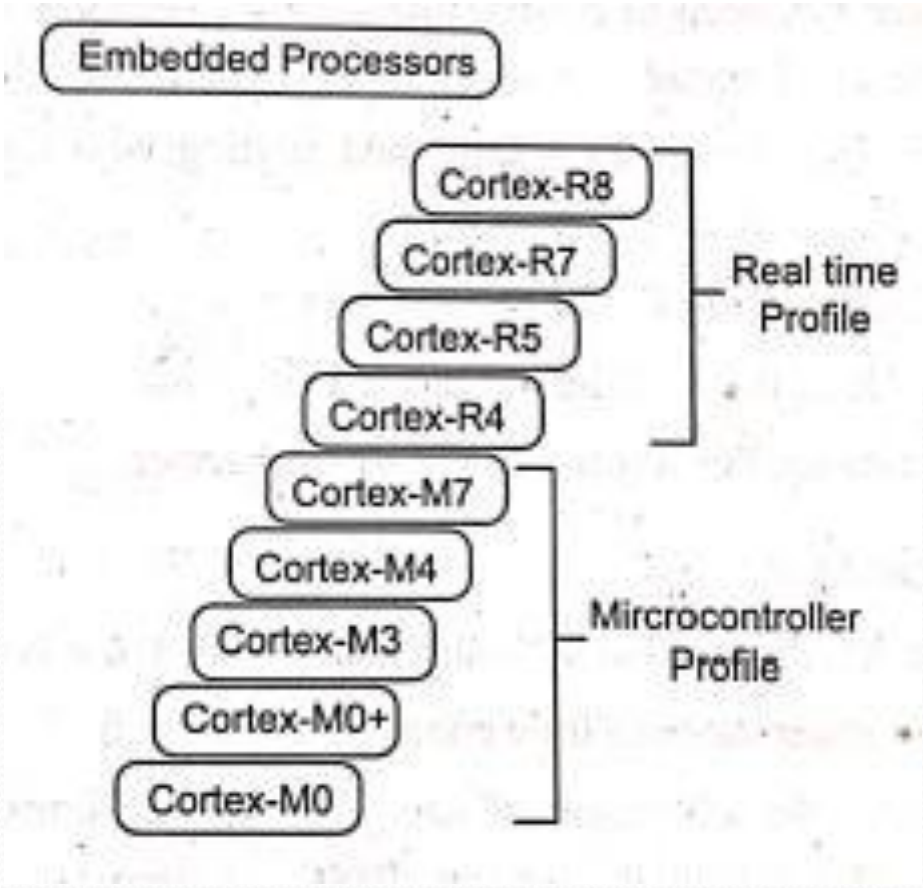
- ARM1136J-S, announced in 2003 was designed for high performance and power efficient applications
- ARM1136J-S was the first processor to execute architecture ARMv6 instructions
- It has eight pipeline stages with load and store arithmetic pipeline.
- ARMv6 instruction are single instruction with multiple data extensions for media processing.

- 2.2 ARM PROCESSORS
- ARM Processor can be divided into three types
  - ARM classic processor
  - ARM Embedded Processor
  - ARM Application processor

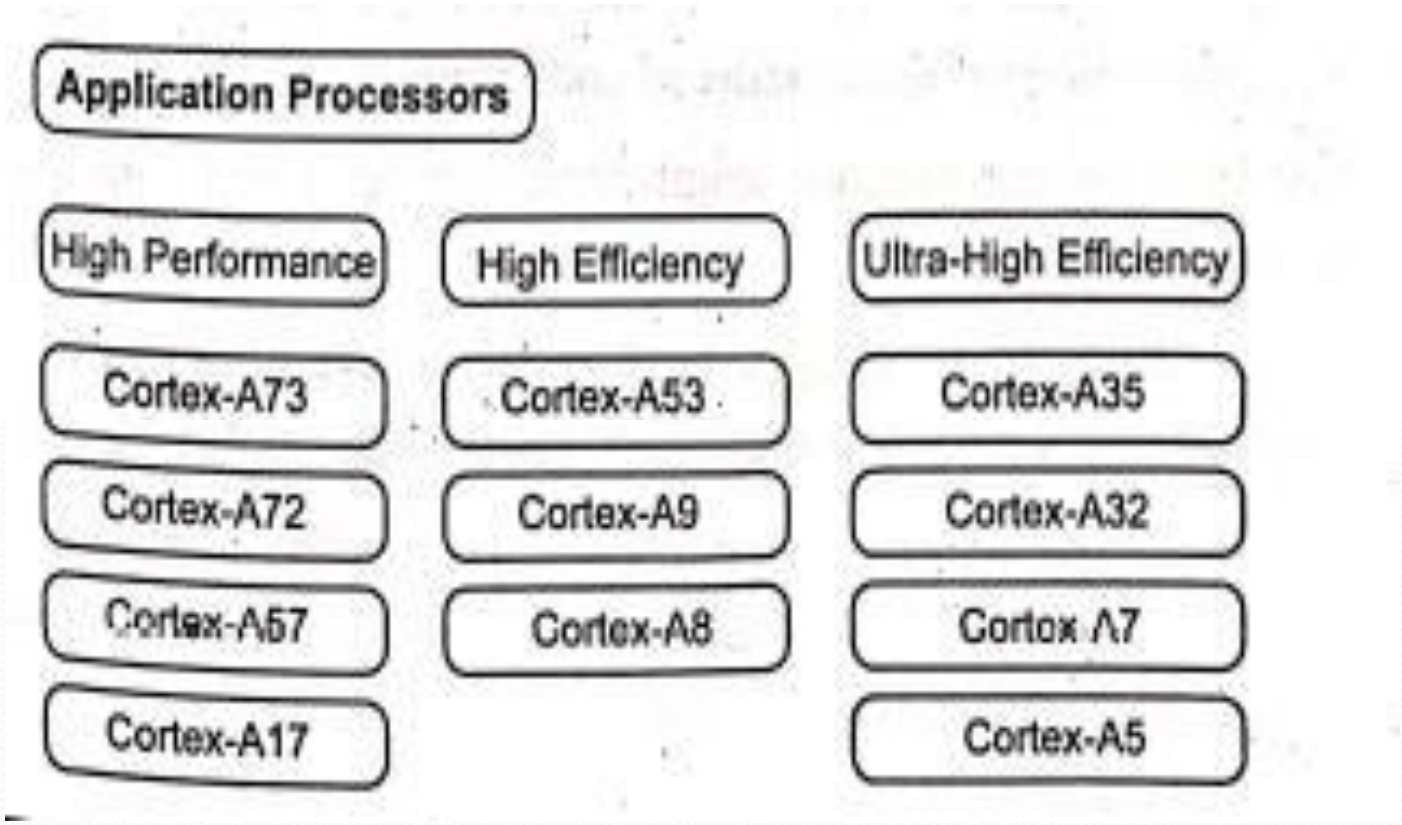
ARM classic processor



ARM Embedded Processor



ARM Application processor

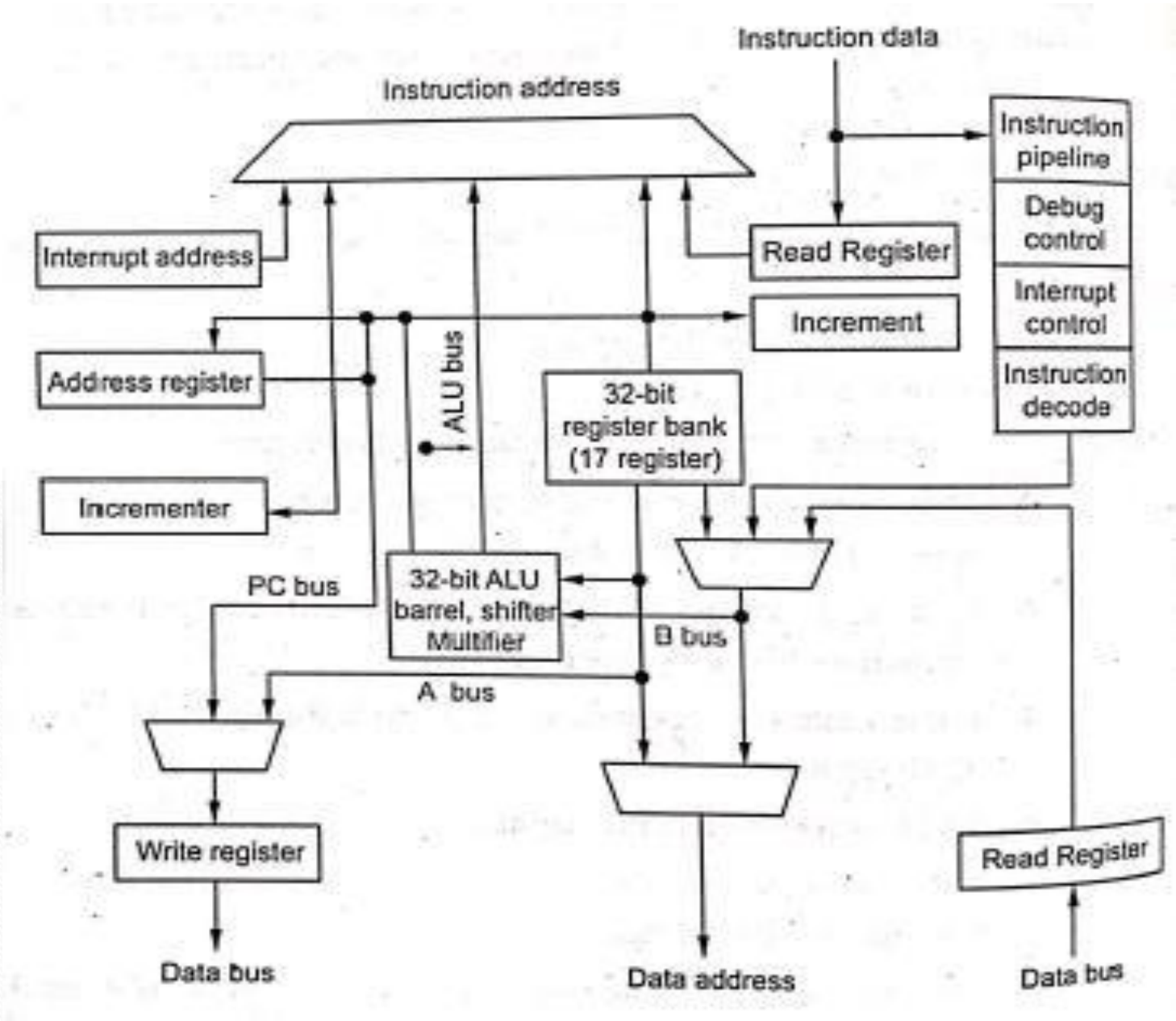


ARM ARCHITECTURE

- The architecture has evolved over time, and starting with cortex series of cores, three profiles are,
- Application Profile Cortex- A series
- Real time profile- Cortex- R series
- Microcontroller profile-Cortex –M series

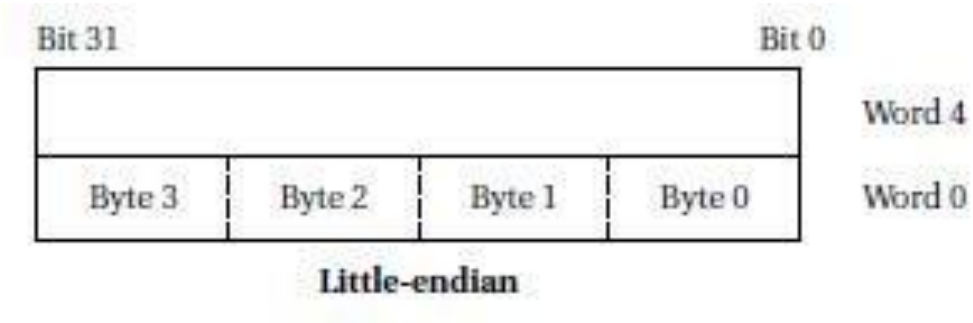
Arm Features

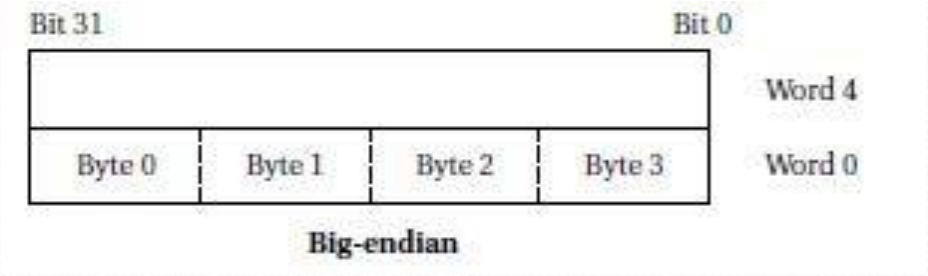
- A load-store architecture,
- Fixed-length 32-bit instructions
- 3-Address instruction formats.


Instruction data
Instruction address
Instruction pipeline
Debug control
Interrupt control
Instruction decode
Interrupt address
Read Register
Increment
Address register
ALU bus
32-bit register bank (17 register)
Incrementer
PC bus
32-bit ALU barrel, shifter Multiplier
B bus
A bus
Write register
Read Register
Data bus
Data address
Data bus

- It has 32 bit architecture but it supports to 16bit and 8 bit data types also
- A wide choice of development tools and simulation models for leading EDA (Electronic Design Automation) environments and excellent debug support
- ARM uses a Intelligent Memory Manager (IEM). It implements advanced algorithms to optimally balance processor workload and power consumption.IEM work with operating system and mobile OS
- ARM uses AHB (AMBA Advanced High performance Bus) interface. AMBA is open source specification for on chip interconnection
-

Byte organizations with an ARM word





ARM ARCHITECTURE

- ARM core is functional units connected by data buses.
- Arrow represents the flow of data.
- Lines represent buses.
- Boxes represent either operation unit or storage area
- Design of ARM is simple and Programmer's design.
- Power Saving design module.
- Flexible design for different application with simple changes
- Instruction Pipeline and Read Data Register are 32 bit

- ARM instructions have two registers:
  - Rm, Rn- source register
  - Rd-destination register.
- Address bus line A(31:0) and data in lines DATA (31:0) to store the data into the register.
- Address Register holds the address of next instruction / data to be fetched
- Address Incrementer the address register value to appropriate amount to point the next instruction/ data
- It contains 31 Register bank, each register are 32 bit registers and also contains 6 status registers each of 32 bits

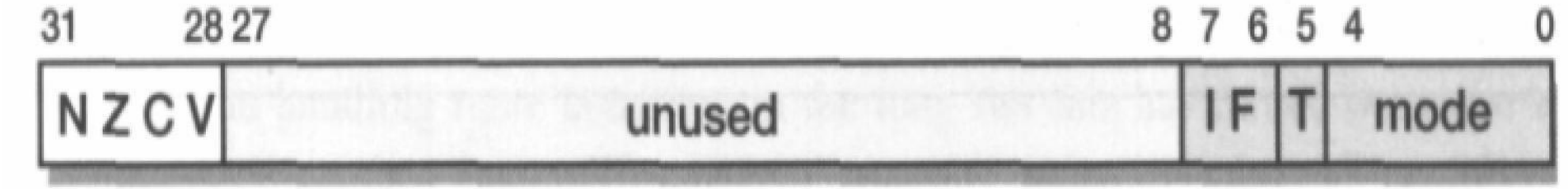
CPU Modes of ARM:

- **User mode:** It is used for programs and applications. It is a only non privileged mode.
- **System Mode:** It is a special version of user mode. It allows the full read write access to the CPSR.
- **Supervisor Mode:** it is privileged mode it enters whenever the processor get reset or SWI instruction is executed. In this mode OS kernel operates in.
- **Abort Mode:** It occurs when there is a failed attempt to access the memory. This mode is entered when prefetch abort and data abort exception occurs.

- **Undefined mode:** it is used when the processor encountered an instruction that is undefined or not supported by the implementation. It is a privileged mode.
- **Interrupt Mode :** It is a privileged mode. When the processor accepts the IRQ it occurs.
- **Fast Interrupt Mode :** It is a privileged mode. When the processor accepts the IRQ it occurs.
- **HYP Mode:** This mode introduced in the ARMV-7A fir cortex- A15 processor to providing hardware virtualization support.

The Current Program Status Register (CPSR)

- It gives the status of ALU result for every execution
- The CPSR is used in user-level programs to store the condition code bits.
- Example, to record the result of a comparison operation and to control whether or not a conditional branch is taken



- **N: Negative**; the last ALU operation which changed the flags produced a **negative result**
- **Z: Zero**; the last ALU operation which changed the flags produced a **zero result** (every bit of the 32-bit result was zero).
- **C: Carry**; the last ALU operation which changed the flags **generated a carry-out**, either as a result of an arithmetic operation in the ALU or from the shifter.
- **V: oVerflow**; the last arithmetic ALU operation which changed the flags **generated an overflow** into the sign bit.

ARM Data Instruction

For arithmetic	ADD	Add
	ADC	Add with carry
	SUB	Subtract
	SBC	Subtract with carry
	RSB	Reverse subtract

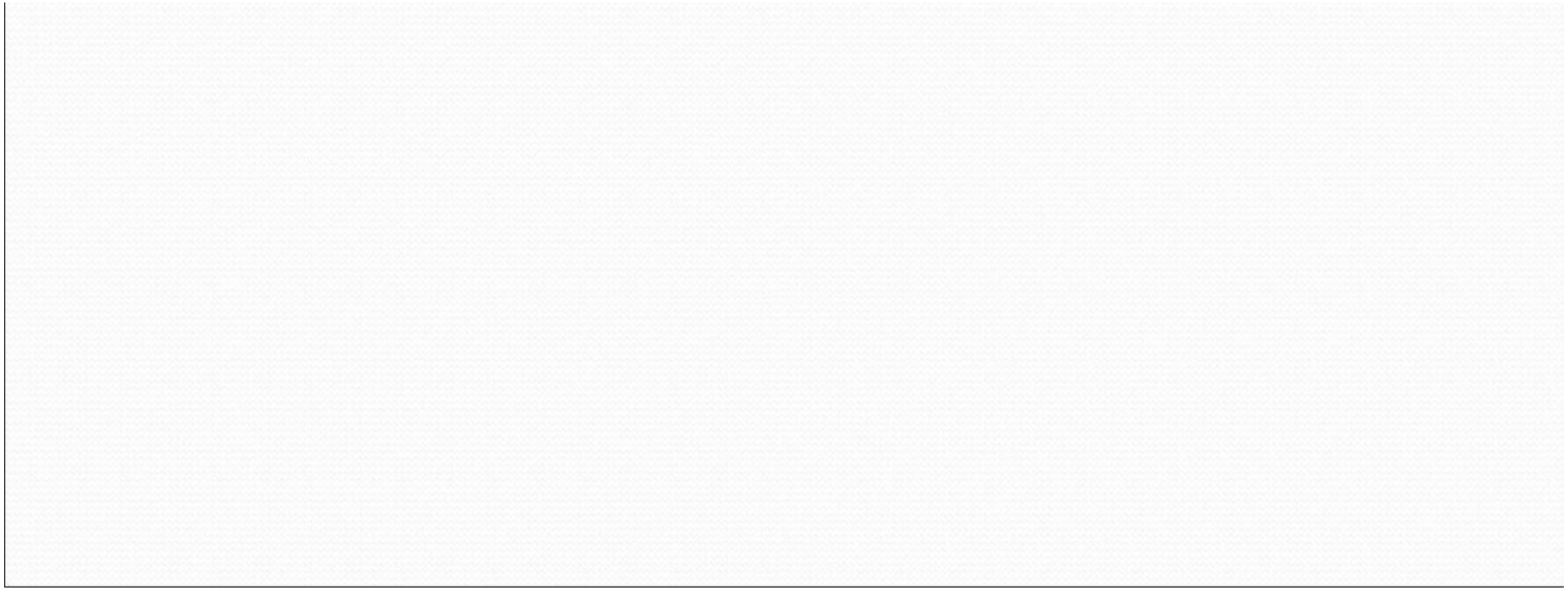
	RSC	Reverse subtract with carry
	MUL	Multiply
	MLA	Multiply and accumulate
For logical	AND	Bit wise and
	ORR	Bit wise or
	EOR	Bit wise exclusive or
	BIC	Bit clear
For shift/rotate	LSL	Logical shift left
	LSR	Logical shift right
	ASL	Arithmetic shift left
	ASR	Arithmetic shift right
	ROR	Rotate right
	RRX	Rotate right extended with C
ARM Comparison Instructions	CMP	Compare
	CMN	Negated compare
	TST	Bit wise test
	TEQ	Bit wise negated test
ARM Move Instructions	MOV	Move
	MVN	Move negated
ARM Load Store Instructions and	LDR	Load
	STR	Store
	LDRSH	Load Half Word

Pseudo Operations	STRH	Store Half Word
	LDRSH	Load Half Word Signed
	LDRB	Load byte
	STRB	Store byte
	ADR	Set register to address

- **Example Program:**
- int a,b,c, X;
- X= a+b-c;

```
ADR r4, a     : get address for a
LDR r0, [r4]  : get value of a
ADR r4, b     : get address for b, reusing r4
LDR r1, [r4]  : load value of b
ADD r3, r0, r1 : set intermediate result for X to a + b
ADR , r4, c   : get address for C
LDR ,r2, [r4] : get value of C
SUB r3, r3, r2 : complete computation of X
ADR r4, X     : get address for X
STR r3, [r4]  : store X at proper location
```

ARM INSTRUCTION SET



Types of instruction set

- Data Processing Instructions
- Branch Instructions
- Load Store Instructions
- Software interrupt Instructions
- Program Status Register Instructions

DATA PROCESSING INSTRUCTIONS:

- Move instruction
- Arithmetic instruction
- Logical instruction
- Comparison instruction
- Multiply instruction

Move instruction

- MOV operand2
- MVN NOT operand2
- MOVS – Update In Status Reg

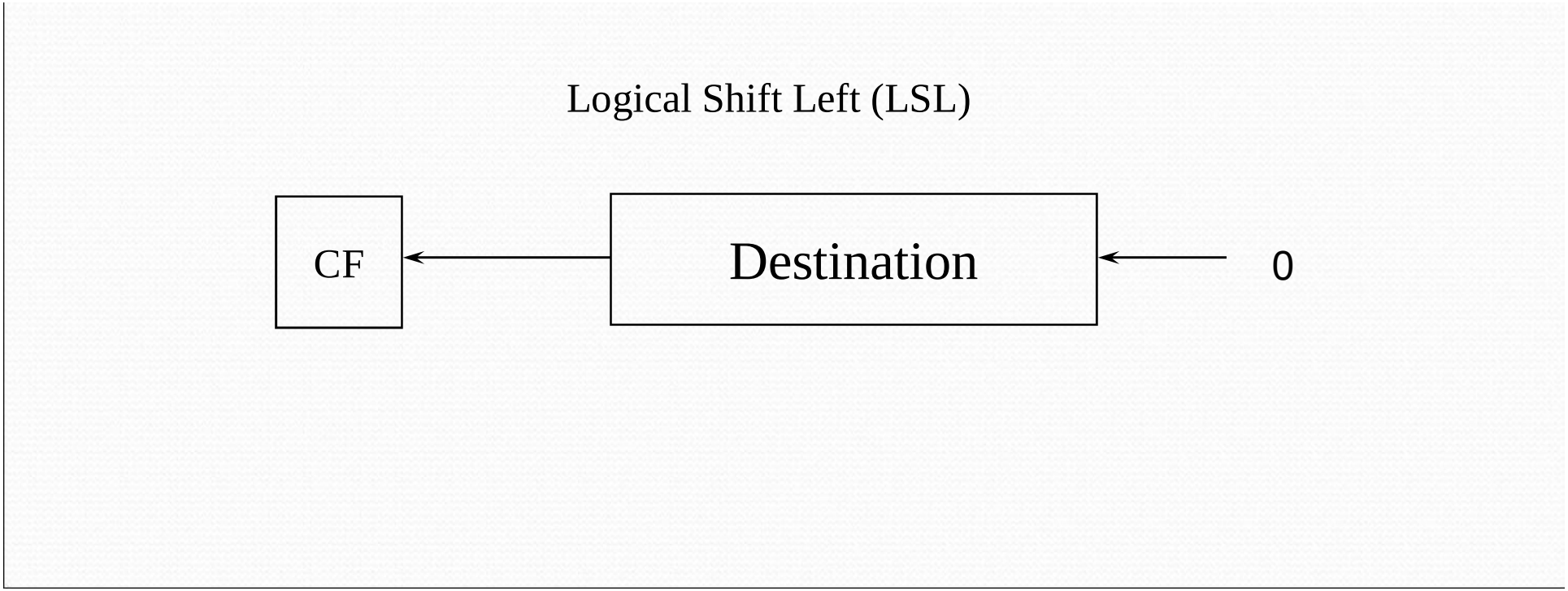
- Syntax:
  - <Operation>{<cond>}{S} Rd, Operand2
- Examples:
  - MOV r0, r1
  - MOVS r2, #10

The Barrel Shifter

- The ARM doesn't have actual shift instructions.
- Instead it has a barrel shifter which provides a mechanism to carry out shifts as part of other instructions.
- **Barrel Shifter - Left Shift**
- Shifts left by the specified amount (multiplies by powers of two)
- e.g.

  LSL #5 = multiply by 32

Barrel Shifter - Left Shift

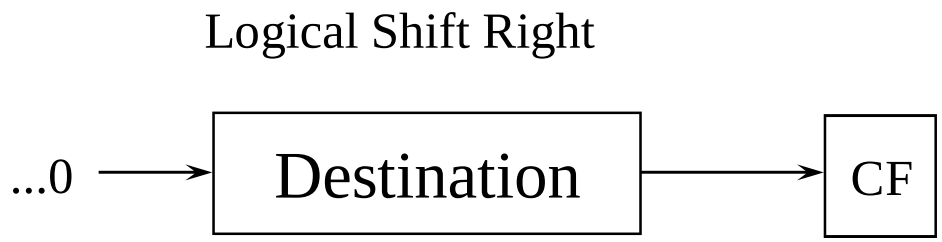


Barrel Shifter - Right Shifts

Logical Shift Right

- Shifts right by the specified amount (divides by powers of two) e.g.

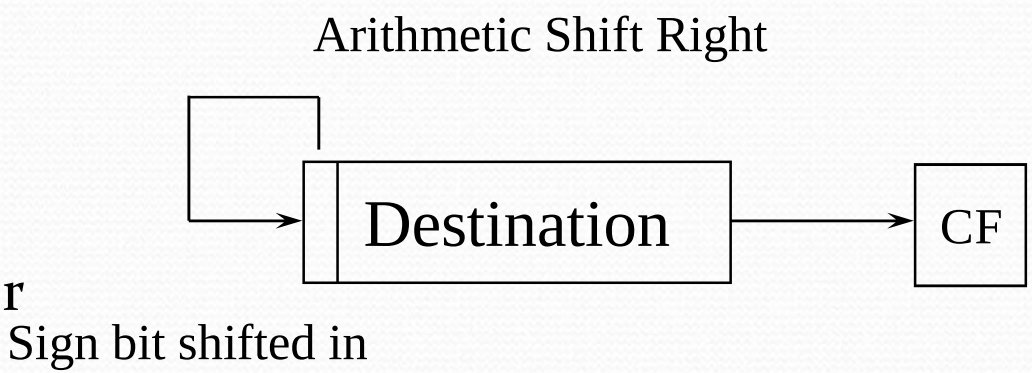
  LSR #5 = divide by 32



Arithmetic Shift Right

- Shifts right (divides by powers of two) and preserves the sign bit, for 2's complement operations. e.g.

  ASR #5 = divide by 32

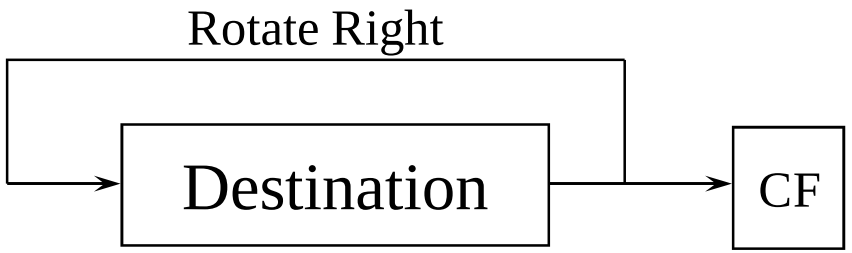


Barrel Shifter - Rotations

Rotate Right (ROR)

- Similar to an ASR but the bits wrap around as they leave the LSB and appear as the MSB.

  e.g. ROR #5

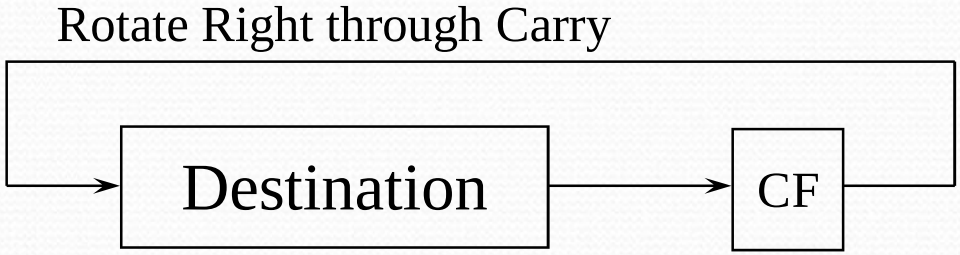


- Note the last bit rotated is also used as the Carry Out.

Rotate Right Extended (RRX)

- This operation uses the CPSR C flag as a 33rd bit.
- Rotates right by 1 bit.

  Encoded as ROR #0.



Arithmetic instruction

ADC	add two 32-bit values and carry	Rd = Rn + N + carry
ADD	add two 32-bit values	Rd = Rn + N
RSB	reverse subtract of two 32-bit values	Rd = N – Rn
RSC	reverse subtract with carry of two 32-bit values	Rd = N – Rn – !(carry flag)
SBC	subtract with carry of two 32-bit values	Rd = Rn – N – !(carry flag)
SUB	subtract two 32-bit values	Rd= Rn – N

example

- ADD r0, r1, r2
  - R0 = R1 + R2
- SUB r5, r3, #10
  - R5 = R3 – 10
- RSB r2, r5, #0xFF00
  - R2 = 0xFF00 – R5

Logical instruction

AND	Logical bit wise AND of two 32-bit values	Rd = Rn & N
ORR	logical bitwise OR of two 32-bit values	Rd = Rn | N
EOR	logical exclusive OR of two 32-bit values	Rd = Rn ^ N
BIC	logical bit clear (AND NOT)	Rd = Rn & ~ N

Comparison instruction

CMN	compare negated	flags set as a result of Rn + N
CMP	compare	flags set as a result of Rn – N
TEQ	test for equality of two 32-bit values	flags set as a result of Rn ∧ N
TST	test bits of a 32-bit value	flags set as a result of Rn & N

Multiply instruction

MLA	multiply and accumulate	Rd = (Rm * Rs) + Rn
MUL	multiply	Rd = Rm * Rs
SMLAL	signed multiply accumulate long	[RdHi, RdLo] = [RdHi, RdLo] + (Rm * Rs)
SMULL	signed multiply long	[RdHi, RdLo] = (Rm * Rs)
UMLAL	unsigned multiply accumulate long	[RdHi, RdLo] = [RdHi, RdLo] + (Rm * Rs)
UMULL	unsigned multiply long	[RdHi, RdLo] + (Rm * Rs)

2. Branch Instructions

B	branch	pc = label
BL	branch with link	pc = label lr = address of the next instruction after the BL
BX	branch exchange	pc = Rm & Oxfffffffe, T = Rm & 1
BLX	branch exchange with link	pc = label, T = 1 pc = Rm & Oxfffffffe, T = Rm & 1 lr = address of the enxt instruction after the BLX

3.Load Store Instructions

Single register transfer

LDR	load word into a register	Rd < - mem32[address]
STR	save byte or word from a register	Rd - > mem32[address]
LORB	load byte into a register	Rd < - mem8[address]
STRB	save byte from a register	Rd - > mem8[address]

LDRH	load halfword into a register	Rd < - mem16[address]
STRH	save halfword into a register	Rd - > mem16[address]

LDRSB	load signed byte into a register	Rd < - Sign Extend (mem8[address])
LDRSH	load signed halfword into a register	Rd < - Sign Extend (mem16[address])

Single register load store addressing mode

Index Method	Data	Base address register	Example
Preindex with writeback	mem[base + offset]	base + offset	LDR r0, [r1, #4]!
Preindex	mem[base + offset]	not updated	LDR r0, [r1, #4]
Postindex	mem[base]	base + offset	LDR r0, [r1], #4

Multiple Register Transfer

LDM	load multiple registers	{Rd}*N < - mem 32 [start address + 4*N] optional Rn updated
STM	save multiple registers	{Rd}*N -> mem32[start address + 4*N] optional Rn updated

Swap instruction

SWP	swap a word between memory and a register	tmp = mem32[Rn] mem32[Rn] = Rm Rd = tmp
SWPB	swap a byte between memory and a register	tmp = mem8[Rn] mem8[Rn] = Rm Rd = tmp

Software interrupt Instructions

SWI	software interrupt	lr_svc = address of instruction following the SWI spsr_svc = cpsr pc=vectors + 0 × 8 cpsr mode = SVC cpsrI = 1 (mask IRQ interrupts)

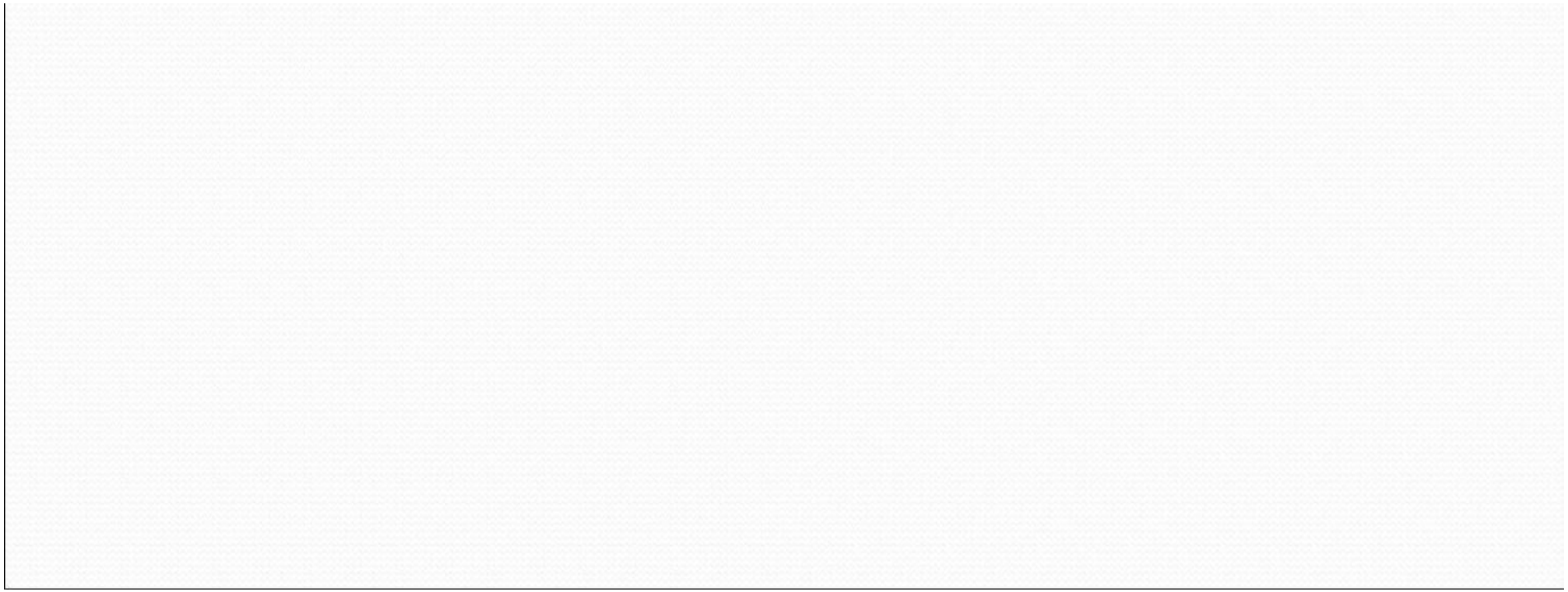
Program Status Register Instructions

MRS	copy program status register to a general-purpose register	Rd= psr
MSR	move a general-purpose register to a program status register	psr[field] = Rm
MSR	move an immediate value to a program status register	psr[field] = immediate

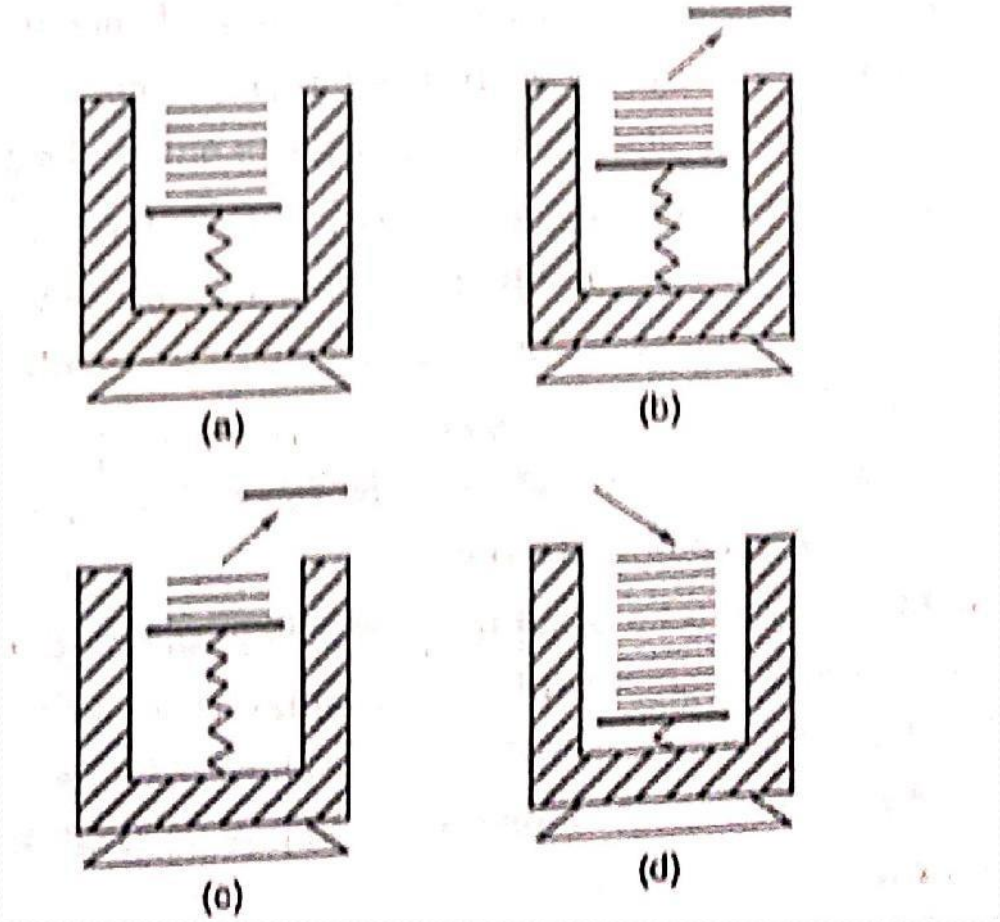
Coprocessor Instruction

CDP	coprocessor data processing – perform an operation in a coprocessor
MRC MCR	coprocessor register transfer – move data to/from coprocessor registers
LDC STC	coprocessor memory transfer – load and store blocks of memory to/from a coprocessor

Stack and subroutine

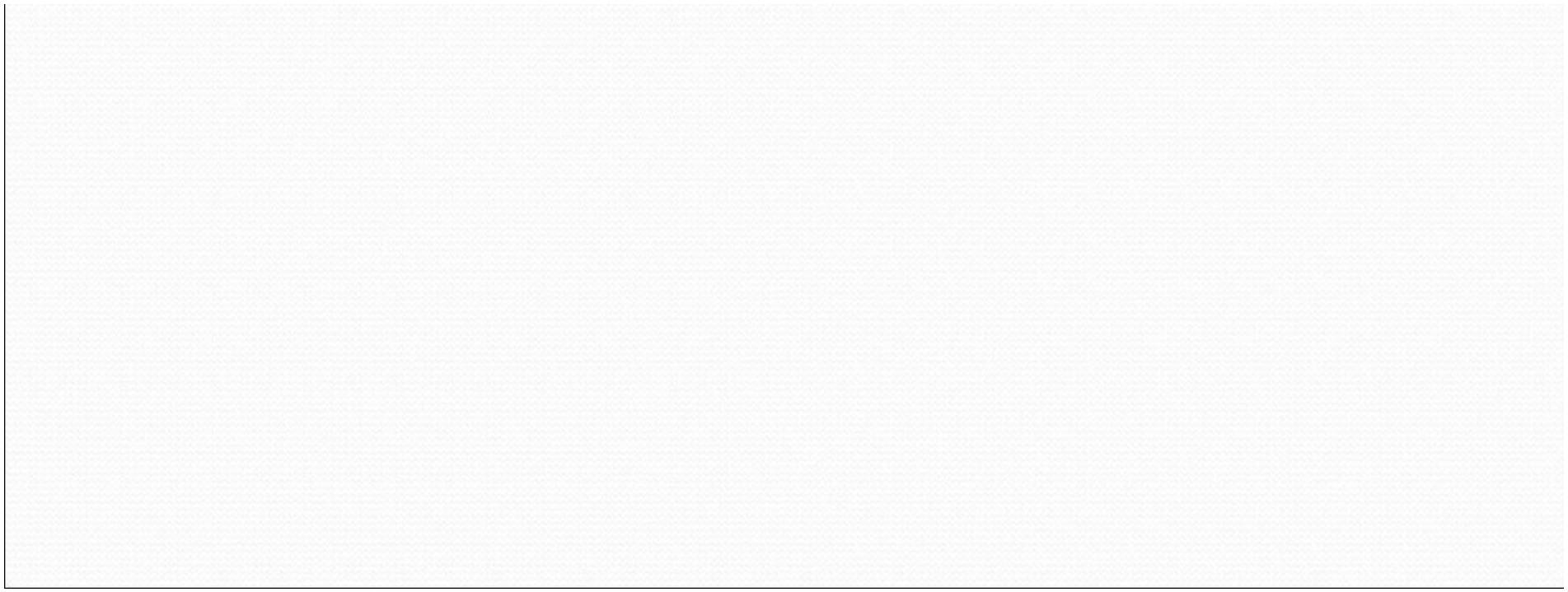


Stack and subroutine



```
        BL      subrout         ; Main program
        .
        .
subrout PUSH    {R0}            ; subroutine
        PUSH    {R1}
        PUSH    {R2}
        PUSH    {R3}
        .
        .   SUBROUTINE INSTRUCTIONS GO HERE
        .
        .
        POP     {R3}
        POP     {R2}
        POP     {R1}
        POP     {R0}
```

- Calling A subroutine
- Parameter passing
- Software delay



2.6 Features of the LPC 214x family

- The LPC2148 is a 16 bit or 32 bit ARM7 family based microcontroller and available in a small LQFP64 package.
- ISP (in system programming) or IAP (in application programming) using on-chip boot loader software.
- On-chip static RAM is 8 kB-40 kB, on-chip flash memory is 32 kB-512 kB, the wide interface is 128 bit, or accelerator allows 60 MHz high-speed operation.
- It takes 400 milliseconds time for erasing the data in full chip and 1 millisecond time for 256 bytes of programming.

- Embedded Trace interfaces and Embedded ICE RT offers real-time debugging with high-speed tracing of instruction execution and on-chip Real Monitor software.
- It has 2 kB of endpoint RAM and USB 2.0 full speed device controller. Furthermore, this microcontroller offers 8kB on-chip RAM nearby to USB with DMA.
- One or two 10-bit ADCs offer 6 or 14 analogs i/ps with low conversion time as 2.44 µs/ channel.
- Only 10 bit DAC offers changeable analog o/p.
- External event counter/32 bit timers-2, PWM unit, & watchdog.
- Low power RTC (real time clock) & 32 kHz clock input.

- Several serial interfaces like two 16C550 UARTs, two I2C-buses with 400 kbit/s speed.5 volts tolerant quick general purpose Input/output pins in a small LQFP64 package.
- Outside interrupt pins-21.60 MHz of utmost CPU CLK-clock obtainable from the programmable-on-chip phase locked loop by resolving time is 100 µs.
- The incorporated oscillator on the chip will work by an exterior crystal that ranges from 1 MHz-25 MHz
- The modes for power-conserving mainly comprise idle & power down.
- For extra power optimization, there are individual enable or disable of peripheral functions and peripheral CLK scaling.

- **2.7 PERIPHERALS:**

- Embedded systems that interacts with the outside world, needs some peripheral device. A peripheral device performs input and output functions for the chip by connecting to other devices or sensors that are off chip.
- Each peripheral device performs one function from outside of chip. Peripheral range is from simple serial communication to complex 802.11 wireless devices.
- All ARM peripherals are memory mapped. It has set of addressed registers. This address registers used to select the exact peripheral device address

Controllers-Specialized peripherals for higher level functionality. Its two types are,

- Memory controllers.
- Interrupt controllers.

Memory controllers:

- Connect different types of memory to the processor bus.
- On- power-up a memory controller is configured in hardware to allow the certain memory devices to be active.
- Some memory devices must be set up by software.

Interrupt controllers:

- When a peripheral device requires a attention it raises the interrupt to the processor.
- The interrupt controller provides the programmable governing policy that allows the software to determine which peripheral device can interrupt the processor at specific time. This is done by bits in the interrupt controller register.

Two types of interrupt controllers for ARM:

- The Standard interrupt controller.
- The Vector interrupt controller (VIC).

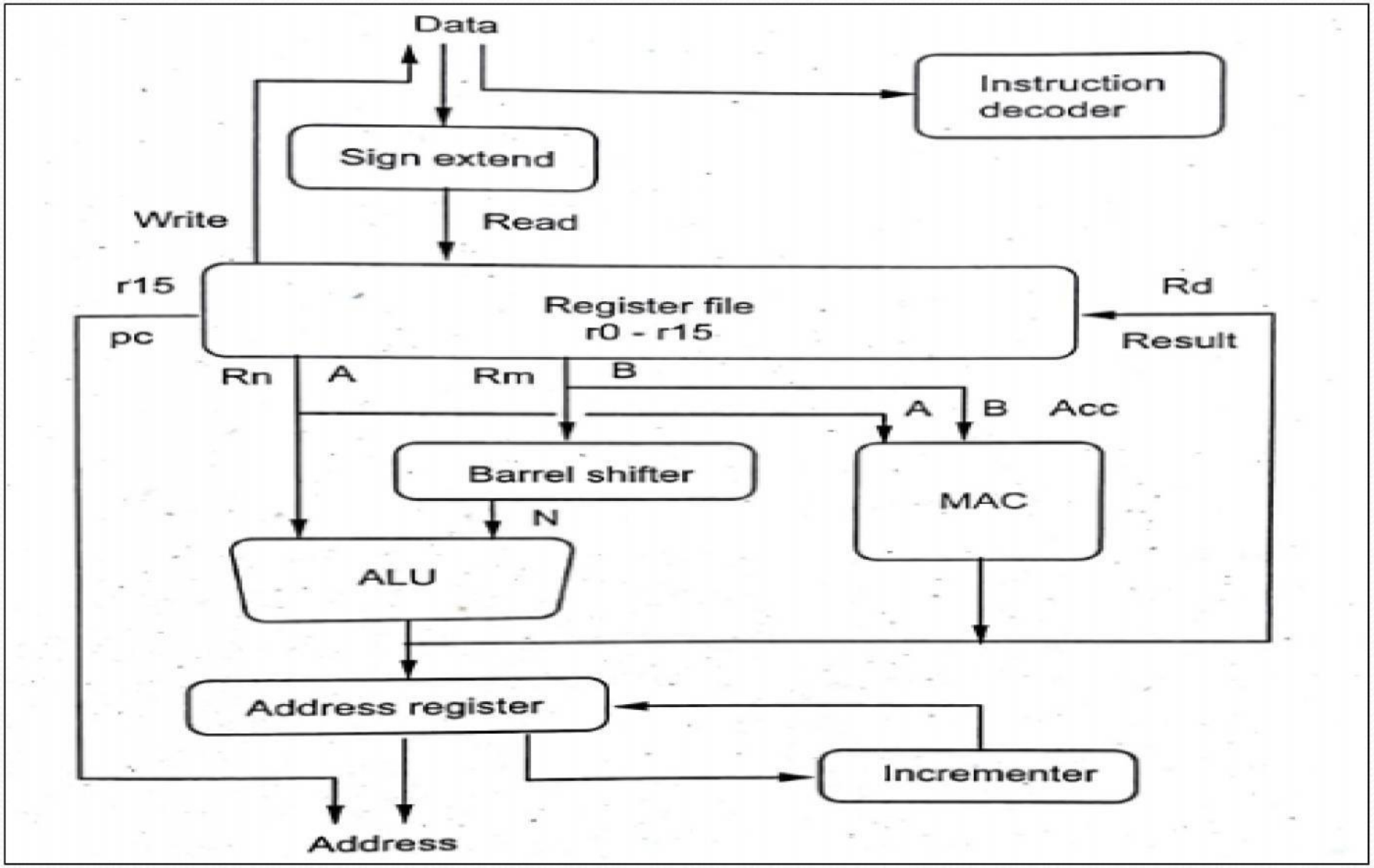
The Standard interrupt controller:

- It sends the interrupt signal to the processor core, when an external device requests servicing.
- It can be programmed to ignore or mask other individual device or set of devices.
- The interrupt handler determines which device requires to servicing by reading a device bitmap register in the interrupt controller.

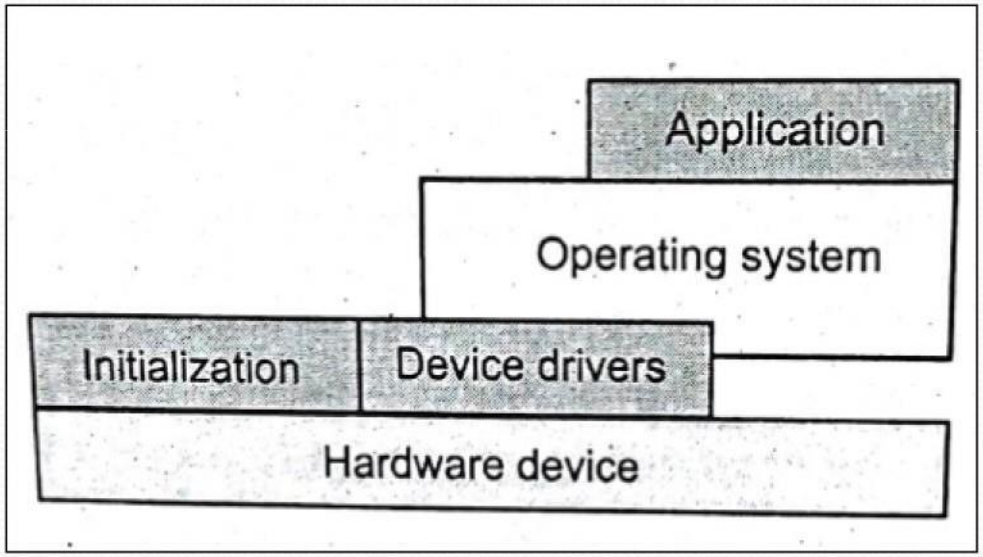
The Vector interrupt controller (VIC):

- It is powerful than Standard interrupt controller. It has prioritizes interrupts. So determination of which device caused the interrupt is simple.
- The VIC only allows an interrupt signal to the core if the new higher priority came than currently executing interrupt.

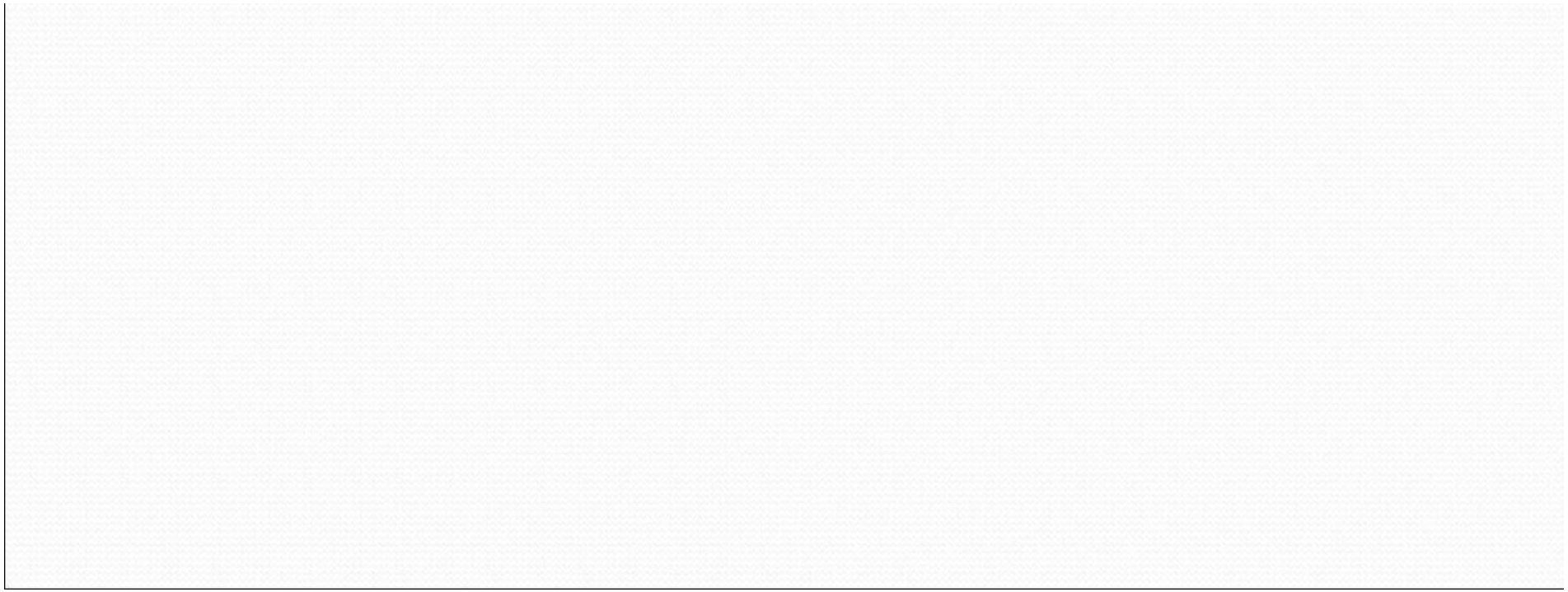
The ARM core data flow model:



Software abstraction layers executing on hardware



The Timer Unit

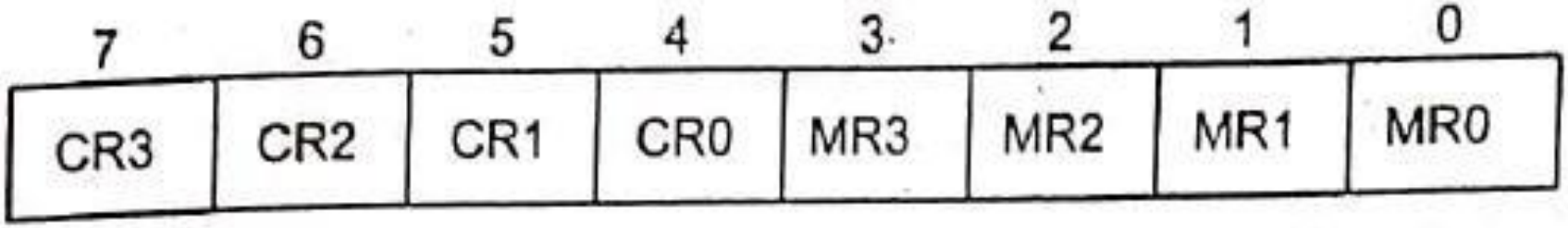


Register Associated with timer in LPC2148

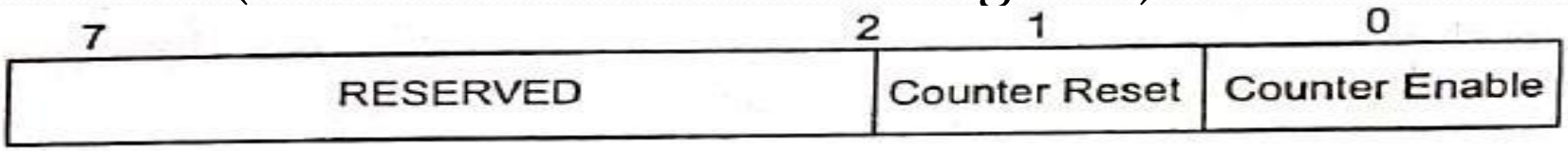
- Prescale register (PR)
- Prescaler Counter register (PC)
- Timer counter register(TC)
- Ttimer control register(TCR)
- Conter control register(CTCR)
- Match control Register (MCR)
- Interrupt Register(IR)

- Timer 0 register

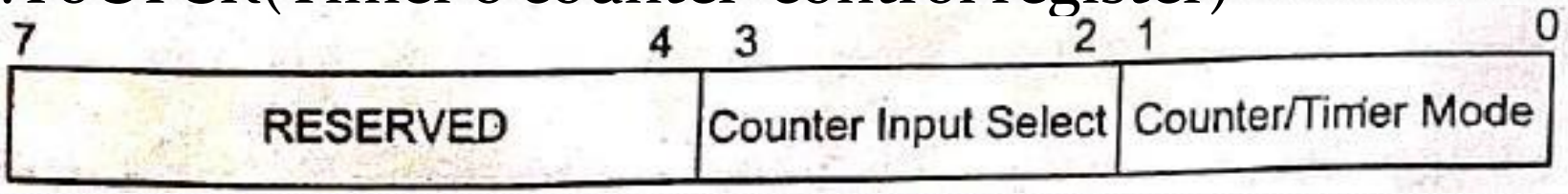
1. ToIR(Timer 0 interrupt Register)



2.ToTCR(Timer 0 Timer Control Register)



3.ToCTCR(Timer 0 counter control register)



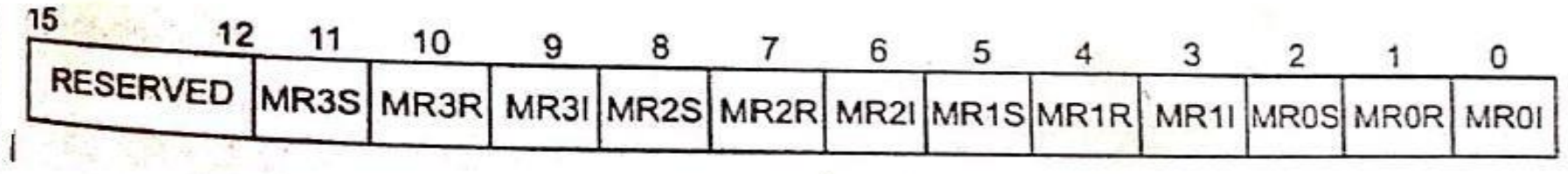
4. ToTC(Timer 0 Timer Counter)

5.ToPR(Timer 0 Prescale Register)

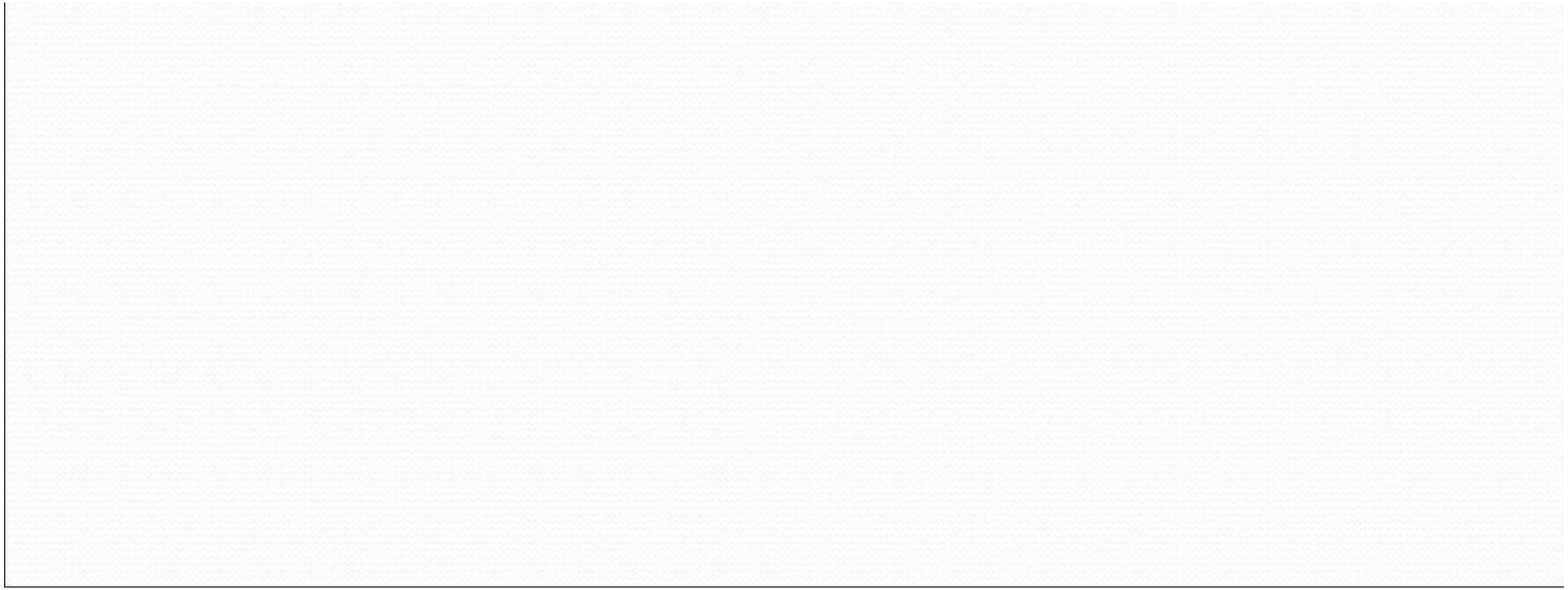
6.ToPC(Timer 0 prescale counter register)

7.ToMR0-ToMR3(Timero Match Register)

8.ToMCR(Timero Match Control Register)

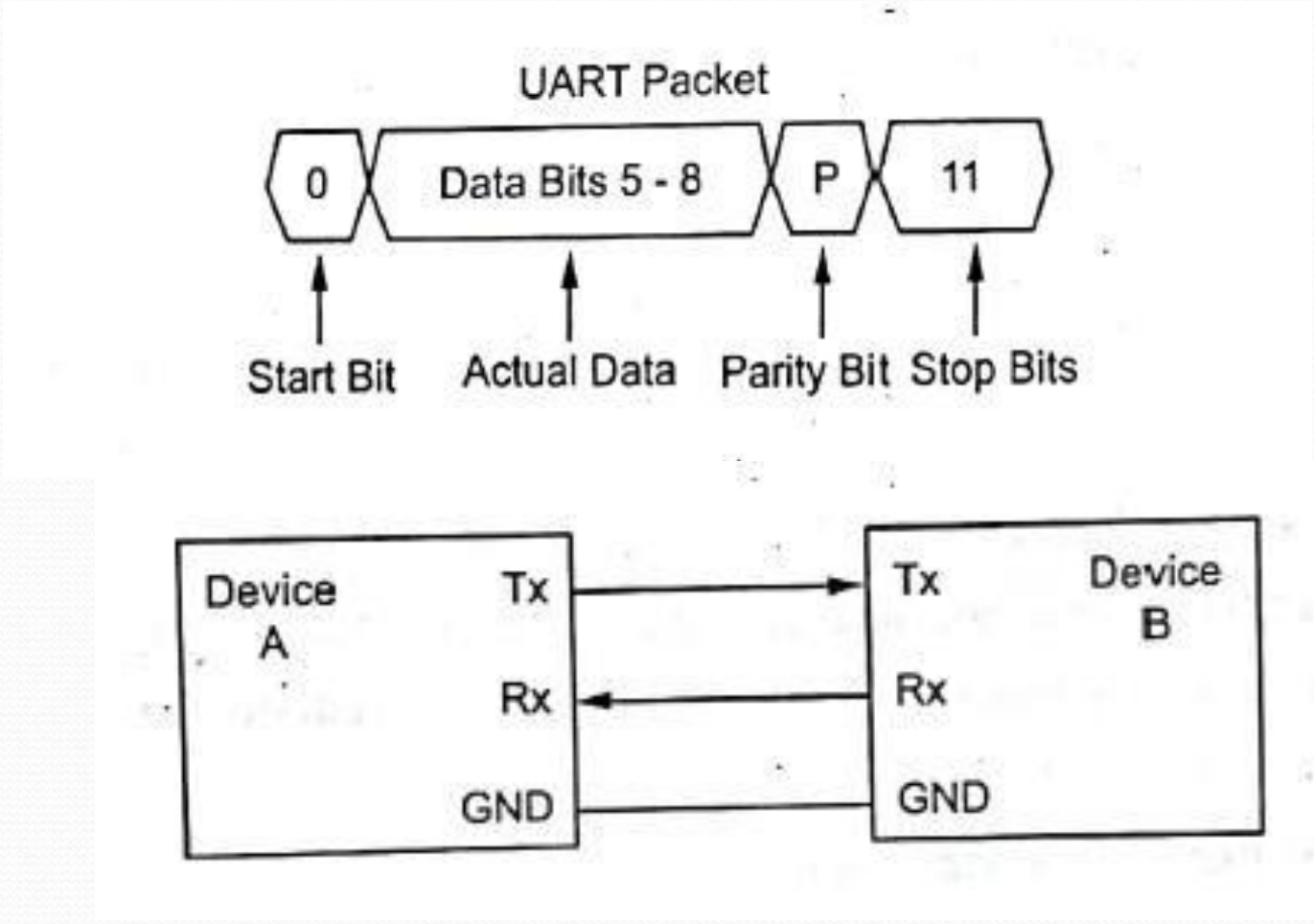


UART



UART

- Universal Asynchronous Receiver/Transmitter



- UART in LPC2148 ARM 7 Micro controller

UART0		UART1	
TXD0	P0.0	TXD1	P0.8
RXD0	P0.1	RXD1	P0.9

Register Associated with UART in LPC2148

- UART0 Receiver Buffer Register(U0RBR)
- UART0 Transmit Holding Register(U0THR)
- UART0 Divisor Latch Register (U0DLL and U0DLM)
  Determine the baud rate generator (U0DLL / U0DLM). (0x00:00x01)
- UART0 Fractional divider register (U0FDR)
  - It is used for prescale for the baud rate
  - Both Multiply and Division can be done in prescale
  - Bit 0 – 3 used for prescale divisor value for baud rate
  - Bit 4 -7 used multiplier value

- UART0 Interrupt Enable Register(U0IER)
  - 0 bit- RBR (Receiver buffer Register)interrupt
  - 1 bit- Interrupt enable register
  - 2 bit- Rx line status register
  - 8 bit – End of auto baud rate interrupt
  - 9 bit- auto baud time out interrupt
- **U0LCR (UART0 Line Control Register)**
- **Bit 1:0 - Word Length Select**

  00 = 5-bit character length
  01 = 6-bit character length
  10 = 7-bit character length
  11 = 8-bit character length

- **Bit 2 - Number of Stop Bits**
  0 = 1 stop bit
  1 = 2 stop bits
- **Bit 3 - Parity Enable**
  0 = Disable parity generation and checking
  1 = Enable parity generation and checking
- **00Bit 5:4 - Parity Select**
  00 = Odd Parity
  01 = Even Parity
  10 = Forced "1" Stick Parity
  11 = Forced "0" Stick Parity
- **Bit 6 - Break Control**
  0= Disable break transmission
  1 = Enable break transmission
- **Bit 7 - Divisor Latch Access Bit (DLAB)**
  0 = Disable access to Divisor Latches
  1 = Enable access to Divisor Latches

U0LSR (UART0 Line Status Register)

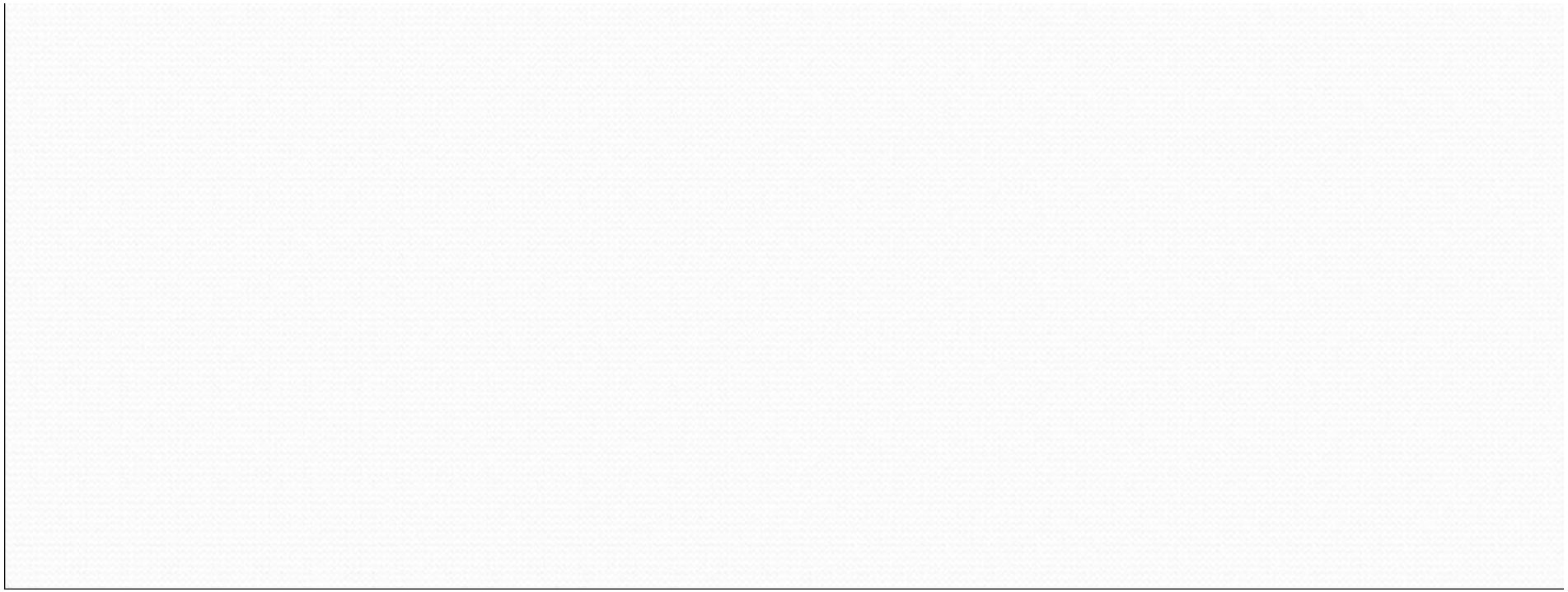
- It provides status information on UART0 RX and TX blocks.
- **Bit 0 - Receiver Data Ready**
  0 = U0RBR is empty
  1 = U0RBR contains valid data
- **Bit 1 - Overrun Error**
  0 = Overrun error status inactive
  1 = Overrun error status active
  This bit is cleared when U0LSR is read.
- **Bit 2 - Parity Error**
  0 = Parity error status inactive
  1 = Parity error status active
  This bit is cleared when U0LSR is read.

- **Bit 3 - Framing Error**
  0 = Framing error status inactive
  1 = Framing error status active
  This bit is cleared when U0LSR is read.
- **Bit 4 - Break Interrupt**
  0 = Break interrupt status inactive
  1 = Break interrupt status active
  This bit is cleared when U0LSR is read.
- **Bit 5 - Transmitter Holding Register Empty**
  0 = U0THR has valid data
  1 = U0THR empty
- **Bit 6 - Transmitter Empty**
  0 = U0THR and/or U0TSR contains valid data
  1 = U0THR and U0TSR empty
- **Bit 7 - Error in RX FIFO (RXFE)**
  0 = U0RBR contains no UART0 RX errors
  1 = U0RBR contains at least one UART0 RX error
  This bit is cleared when U0LSR is read

U0TER (UART0 Transmit Enable Register)

- The U0TER enables implementation of software flow control. When TXEn=1, UART0 transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, UART0 transmission will stop.
- Software implementing software-handshaking can clear this bit when it receives an XOFF character (DC3). Software can set this bit again when it receives an XON (DC1) character.
- **Bit 7 : TXEN**
  0 = Transmission disabled
  1 = Transmission enabled
- If this bit is cleared to 0 while a character is being sent, the transmission of that character is completed, but no further characters are sent until this bit is set again

Block Diagram of ARM9



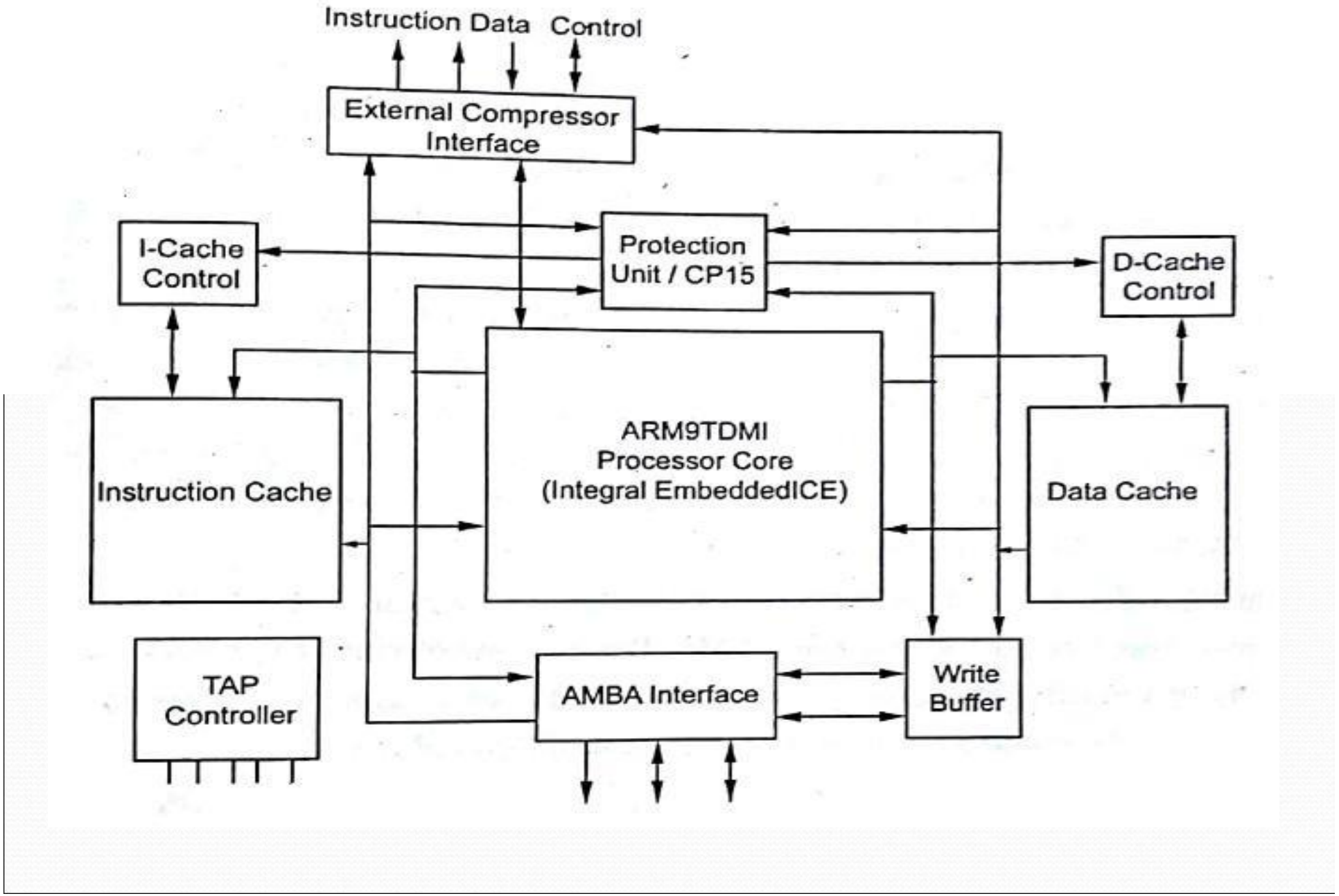
2.12.1. FEATURES OF ARM9

1. **Pipeline Depth:** 5 stage (Fetch, Decode, Execute, Decode, Write)
2. **Operating frequency:** 150 MHz
3. **Power Consumption:** 0.19 mW/MHz
4. **MIPS/MHz:** 1.1
5. **Architecture used:** Harvard
6. **MMU/MPU:** Present
7. **Cache Memory:** Present (separate 16k/8k)
8. **ARM/ Thumb Instruction:** Support both
9. **ISA (Instruction Set Architecture):** V5T(ARM926EJ-S)
10. 31 (32-Bit size) Registers
11. 32-bit ALU & Barrel Shifter
12. Enhanced 32- bit MAC block
13. Memory Controller

Memory operations are controlled by MMU or MPU

1. **MMU:**
   - Provides Virtual Memory Support
   - Fast Context Switching Extensions
2. **MPU:**
   - Enables memory protection & bounding
   - Sand – boxing of applications

14. Flexible Cache Design (sizes can be 4KB to 128KB)
15. Flexible Core Design
16. DSP Enhancements: (very important)
17. Single cycle 32 ×16 multiplier Implementation
18. Speed up all the multiply instructions
19. New 32 ×16 & 16 ×16 multiply instructions
20. Allows independent access to 16 bit halves of registers


Instruction Data Control
External Compressor Interface
I-Cache Control
Protection Unit / CP15
D-Cache Control
Instruction Cache
ARM9TDMI Processor Core (Integral EmbeddedICE)
Data Cache
TAP Controller
AMBA Interface
Write Buffer

ARM9TDMI

- ARM920T with 16 KB each of I/D cache and an MMU
- ARM922T with 8 KB each of I/D cache and an MMU
- ARM940T with cache and a Memory Protection Unit (MPU)

ARM940T Cached Processor

- Firstly, it allows the processor to operate at its maximum frequency since memory accesses are to the local, high performance cache.
- Secondly, since main memory is accessed infrequently, system power is reduced. Also, the main memory system may now be used for other tasks, such as DMA, while the processor is executing from its caches.

Comparision between ARM9TDMI and ARM7TDMI

Instruction	% taken	% Skipped	ARM7TDMI	ARM9TDMI
Data processing	49	4	1	1
Data processing with PC	3	0	3	3
Branch/Branch with link	11	4	3	3
Load register	14	1	3	1-2
Store register	8	1	2	1
Load multiple registers	1	0	7	5
Store multiple registers	2	0	7	6
CPI			1.9	1.5

Pipeline Process

...maximum operating frequency.

ARM7TDMI Pipeline Operation

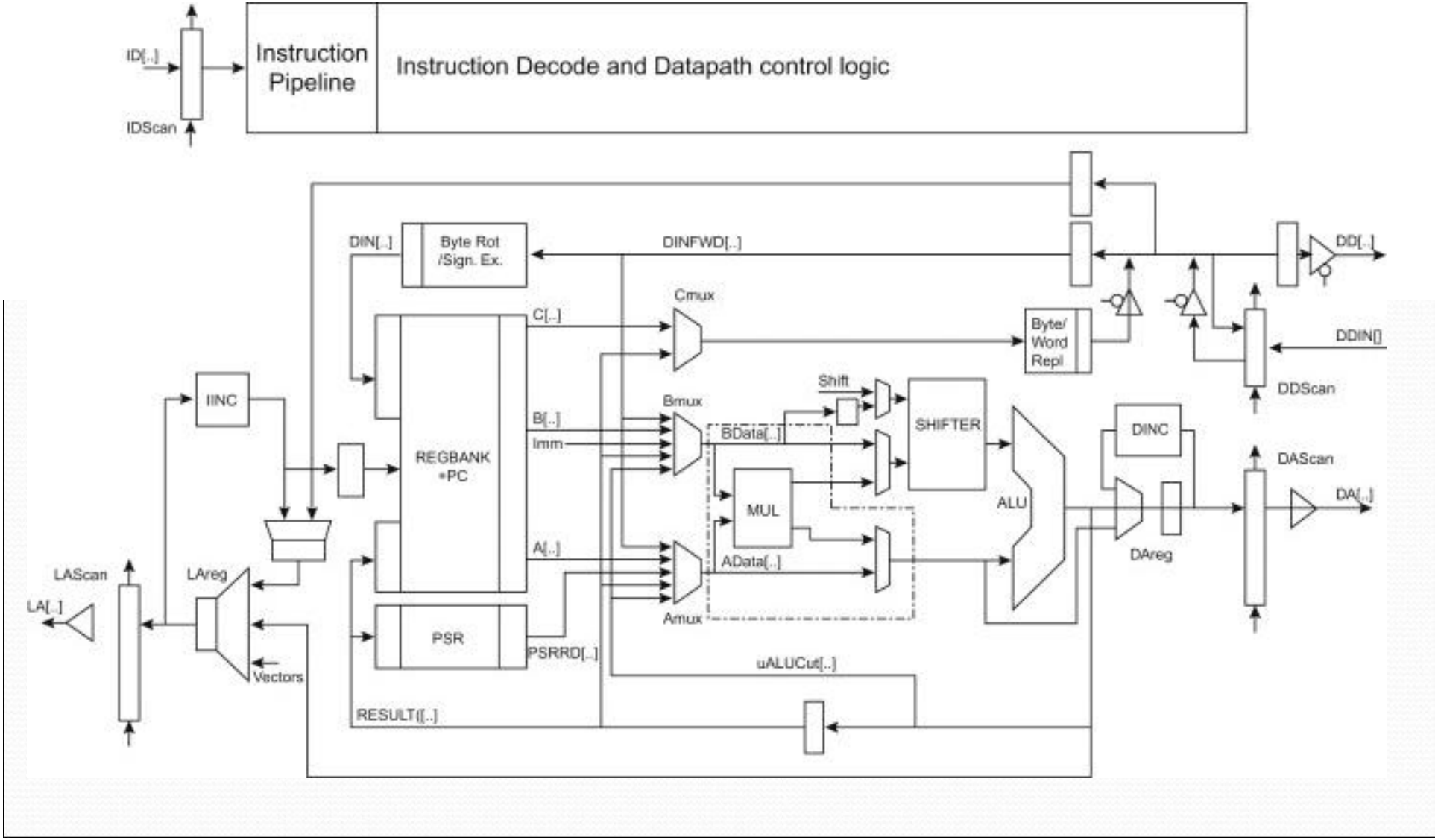
Fetch	Decode	Execute
Instruction Fetch	Convert Thumb to ARM Main Decode Register Address Decode	Register Read Shifter ALU Writeback

ARM9TDMI Pipeline Operation

Fetch	Decode	Execute	Memory	Writeback
Instruction Fetch	Reg. Address Decode Register Read Thumb Decode Reg. Address Decode Register Read	Shifter ALU	Memory Data access	ALU Result and / or Load data Writeback

Fig. 2.30 ARM7TDMI...

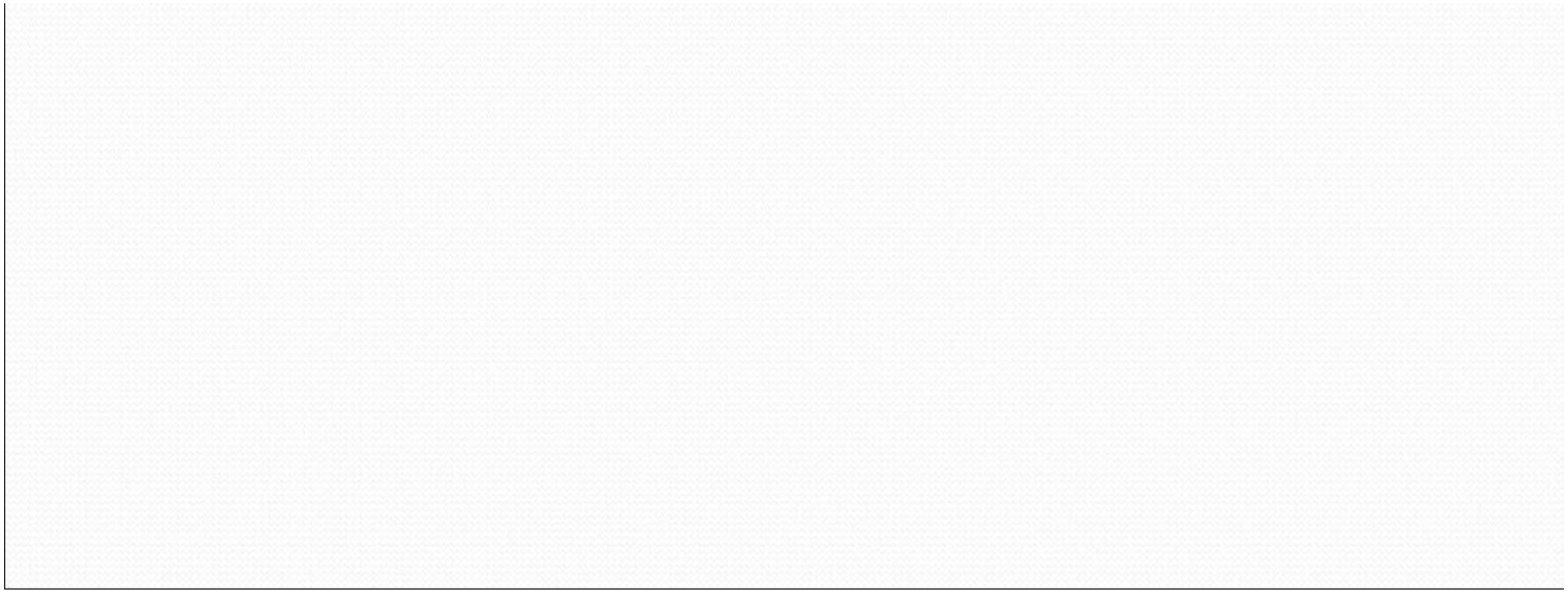
DATA FLOW


ID[..]
IDScan
Instruction Pipeline
Instruction Decode and Datapath control logic
DIN[..]
Byte Rot /Sign. Ex.
DINFWD[..]
DD[..]
Cmux
C[..]
Byte/ Word Repl
DDIN[]
DDScan
IINC
Bmux
Shift
B[..]
Imm
BData[..]
SHIFTER
DINC
DAScan
REGBANK *PC
MUL
ALU
DA[..]
A[..]
AData[..]
DAreg
LAScan
LAreg
LA[..]
Amux
PSR
PSRRD[..]
uALUCut[..]
Vectors
RESULT[..]

COMPARISION SUMMARY

	ARM7TDMI	ARM9TDMI
Area (mm^2,0.35 μm)	2.2	4.15
Transistor count	74k	112k
Pipeline stages	3	5
CPI	1.9	1.5
MIPS/MHz	0.9	1.1
Typical Max Clock rate (0.35 μm)	60	120
Power (mW/MHz @ 3.0V)	1.5	1.8

2.9 Pulse Width Modulation(PWM)



$$\text{Duty Cycle (In \%)} = \frac{T_{on}}{T_{on} + T_{off}} \times 100$$

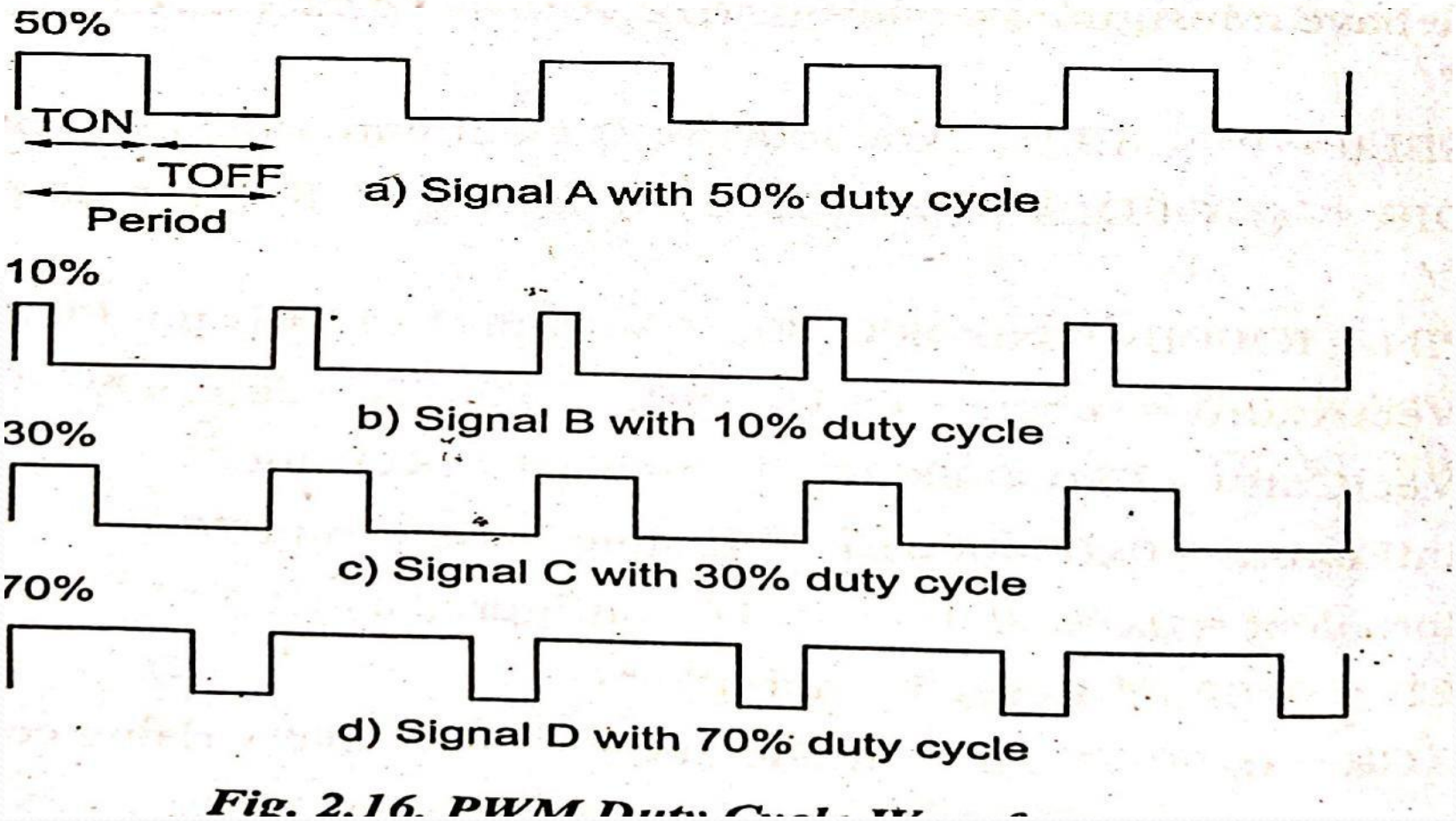
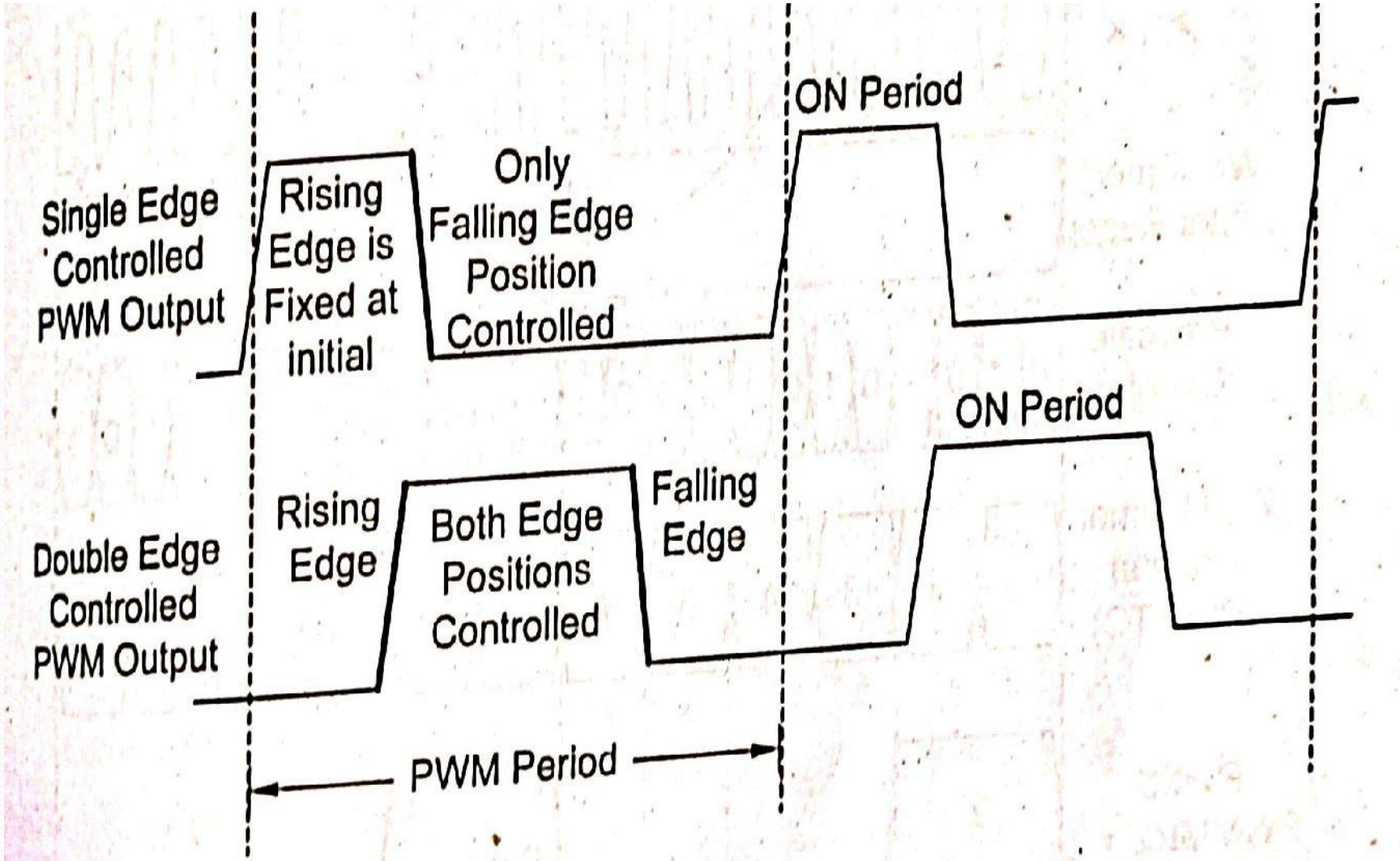


Fig. 2.16. PWM Duty Cycle W...

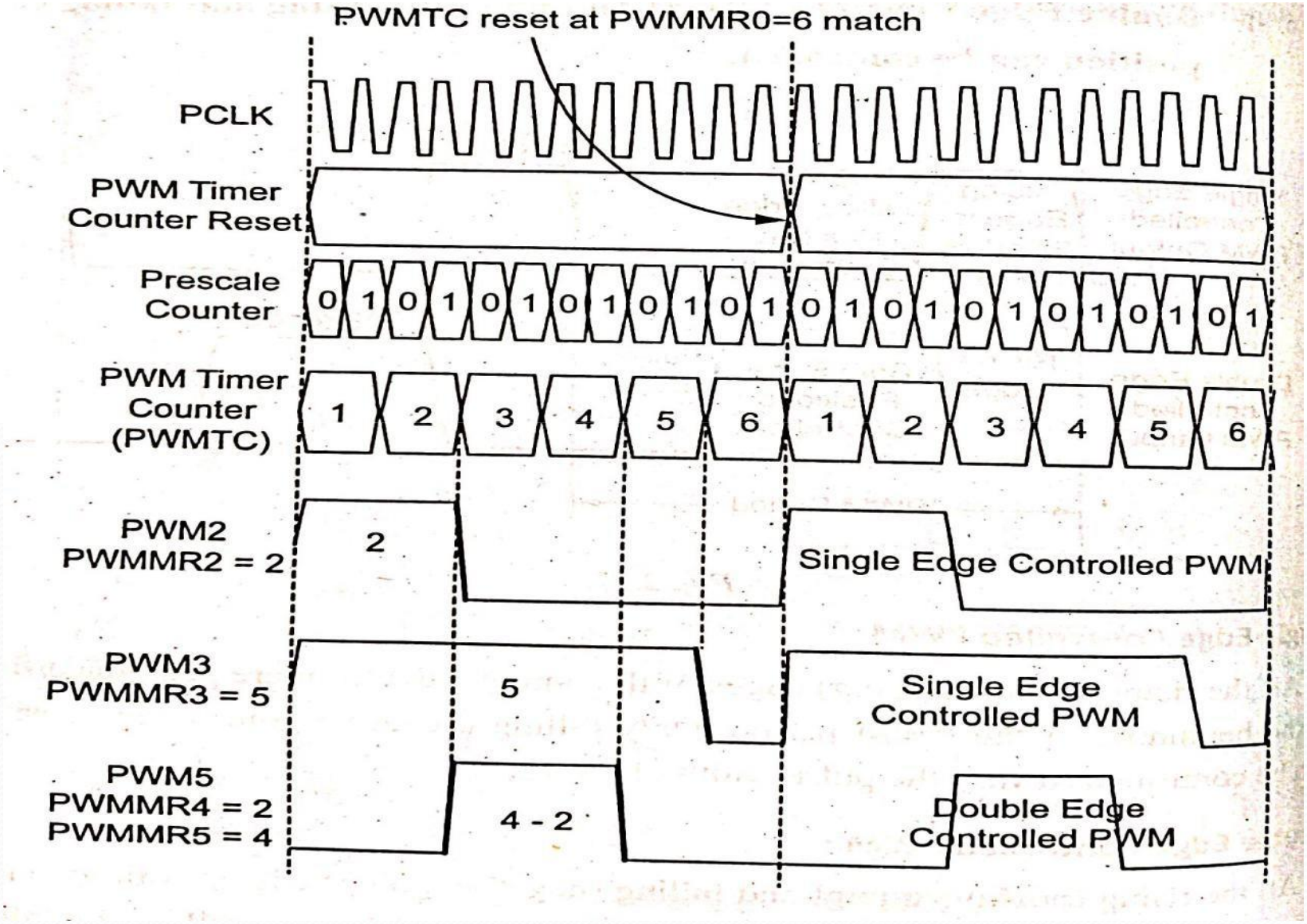

Single Edge
Controlled
PWM Output
Rising
Edge is
Fixed at
initial
Only
Falling Edge
Position
Controlled
ON Period
Double Edge
Controlled
PWM Output
Rising
Edge
Both Edge
Positions
Controlled
Falling
Edge
ON Period
PWM Period

LPC 2148

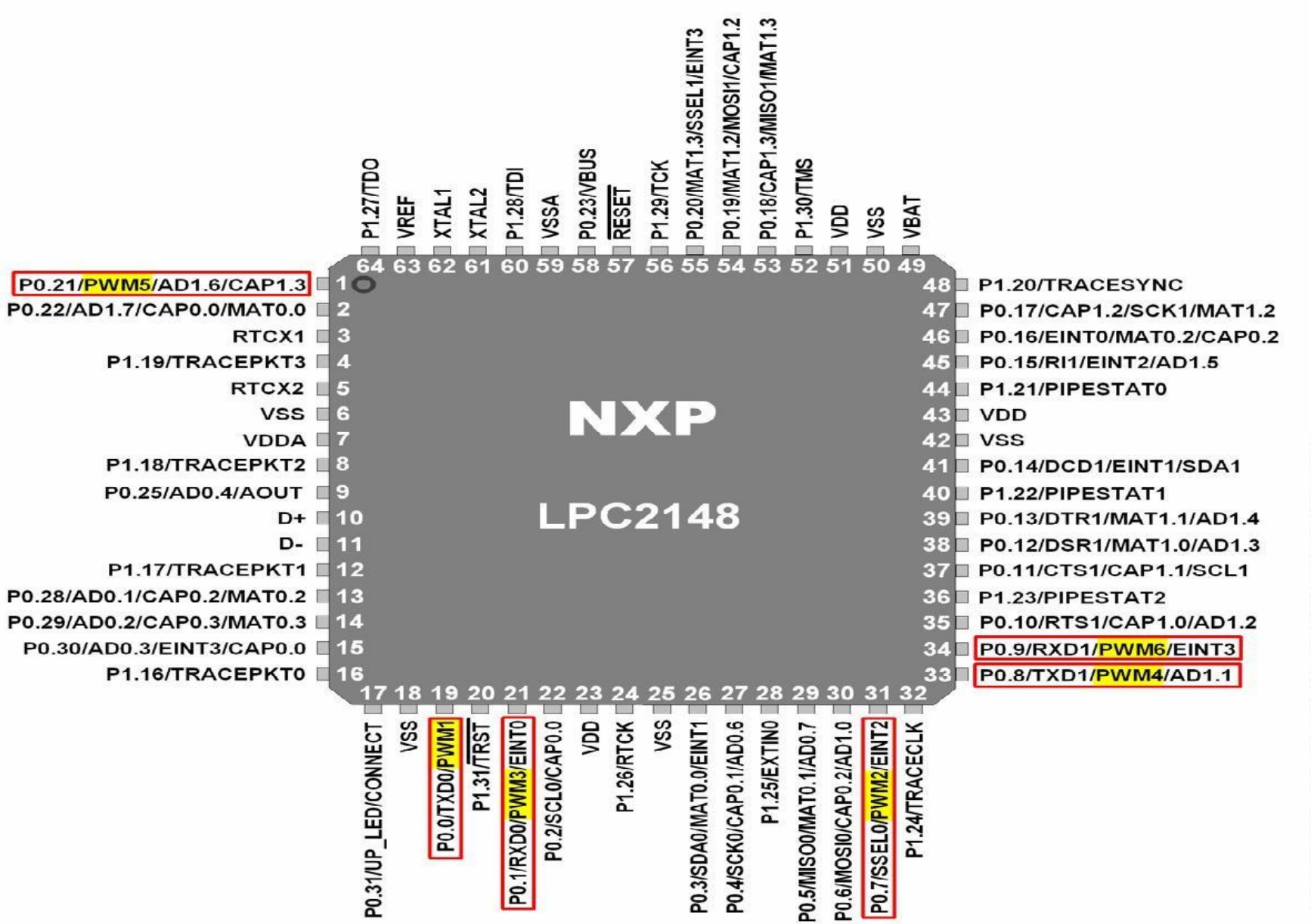
- It consist of 32 timer /counter ie PWMTC
- Counter count the cycles of peripheral clock(PCLK)
- It having 32bit prescale register (PWMPR)
- It having 7 matching register (PWMR0-PWMR06)
- 6 different pwm signal in single edge controlled pwm

or 3 different pwm signal in double edge controlled pwm

- Match register will match and then it will reset the timer/counter or stop.

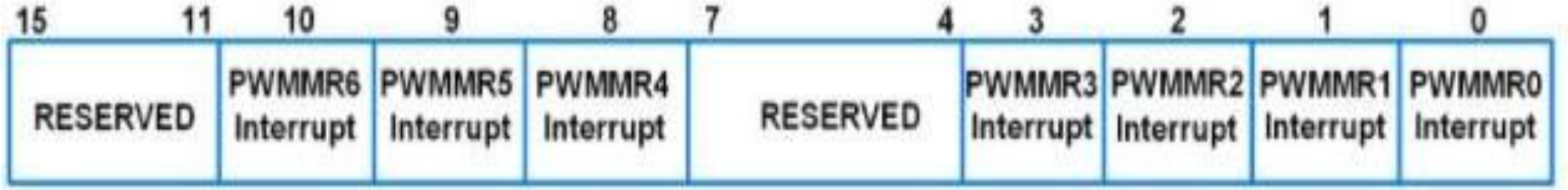

PWMTC reset at PWMMR0=6 match
PCLK
PWM Timer Counter Reset
Prescale Counter
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
PWM Timer Counter (PWMTC)
1 2 3 4 5 6 1 2 3 4 5 6
PWM2 PWMMR2 = 2
2
Single Edge Controlled PWM
PWM3 PWMMR3 = 5
5
Single Edge Controlled PWM
PWM5 PWMMR4 = 2 PWMMR5 = 4
4 - 2
Double Edge Controlled PWM

PWM Channel	Single Edge Controlled		Double Edge Controlled	
	Set by	Reset by	Set by	Reset by
1	Match 0	Match 1	Match 0	Match 1
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2	Match 3
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4	Match 5
6	Match 0	Match 6	Match 5	Match 6


NXP
LPC2148
1 P0.21/PWM5/AD1.6/CAP1.3
2 P0.22/AD1.7/CAP0.0/MAT0.0
3 RTCX1
4 P1.19/TRACEPKT3
5 RTCX2
6 VSS
7 VDDA
8 P1.18/TRACEPKT2
9 P0.25/AD0.4/AOUT
10 D+
11 D-
12 P1.17/TRACEPKT1
13 P0.28/AD0.1/CAP0.2/MAT0.2
14 P0.29/AD0.2/CAP0.3/MAT0.3
15 P0.30/AD0.3/EINT3/CAP0.0
16 P1.16/TRACEPKT0
17 P0.31/UP_LED/CONNECT
18 VSS
19 P0.0/TXD0/PWM1
20 P1.31/TRST
21 P0.1/RXD0/PWM3/EINT0
22 P0.2/SCL0/CAP0.0
23 VDD
24 P1.26/RTCK
25 VSS
26 P0.3/SDA0/MAT0.0/EINT1
27 P0.4/SCK0/CAP0.1/AD0.6
28 P1.25/EXTIN0
29 P0.5/MISO0/MAT0.1/AD0.7
30 P0.6/MOSI0/CAP0.2/AD1.0
31 P0.7/SSEL0/PWM2/EINT2
32 P1.24/TRACECLK
33 P0.8/TXD1/PWM4/AD1.1
34 P0.9/RXD1/PWM6/EINT3
35 P0.10/RTS1/CAP1.0/AD1.2
36 P1.23/PIPESTAT2
37 P0.11/CTS1/CAP1.1/SCL1
38 P0.12/DSR1/MAT1.0/AD1.3
39 P0.13/DTR1/MAT1.1/AD1.4
40 P1.22/PIPESTAT1
41 P0.14/DCD1/EINT1/SDA1
42 VSS
43 VDD
44 P1.21/PIPESTAT0
45 P0.15/RI1/EINT2/AD1.5
46 P0.16/EINT0/MAT0.2/CAP0.2
47 P0.17/CAP1.2/SCK1/MAT1.2
48 P1.20/TRACESYNC
49 VBAT
50 VSS
51 VDD
52 P1.30/TMS
53 P0.18/CAP1.3/MISO1/MAT1.3
54 P0.19/MAT1.2/MOSI1/CAP1.2
55 P0.20/MAT1.3/SSEL1/EINT3
56 P1.29/TCK
57 RESET
58 P0.23/VBUS
59 VSSA
60 P1.28/TDI
61 XTAL2
62 XTAL1
63 VREF
64 P1.27/TDO

PWM Registers

1. PWMIR (PWM Interrupt Register)



- It has 7 interrupt bits corresponding to the 7 PWM match registers.
- If an interrupt is generated, then the corresponding bit in this register becomes HIGH.
- Otherwise the bit will be LOW.
- Writing a 1 to a bit in this register clears that interrupt.
- Writing a 0 has no effect.

2. PWMTCR (PWM Timer Control Register)

- It is an 8-bit register.
- It is used to control the operation of the PWM Timer Counter.
- **Bit 0 – Counter Enable**
  When 1, PWM Timer Counter and Prescale Counter are enabled.
  When 0, the counters are disabled.
- **Bit 1 – Counter Reset**
  When 1, the PWM Timer Counter and PWM Prescale Counter are synchronously reset on next positive edge of PCLK.
  Counter remains reset until this bit is returned to 0.
- **Bit 3 – PWM Enable**
  This bit always needs to be 1 for PWM operation. Otherwise PWM will operate as a normal timer.
  When 1, PWM mode is enabled and the shadow registers operate along with match registers.
  A write to a match register will have no effect as long as corresponding bit in PWMLER is not set.

3. PWMTC (PWM Timer Counter)

- It is a 32-bit register.
- It is incremented when the PWM Prescale Counter (PWMPC) reaches its terminal count.

4. PWMPR (PWM Prescale Register)

- It is a 32-bit register.
- It holds the maximum value of the Prescale Counter.

5. PWMPC (PWM Prescale Counter)

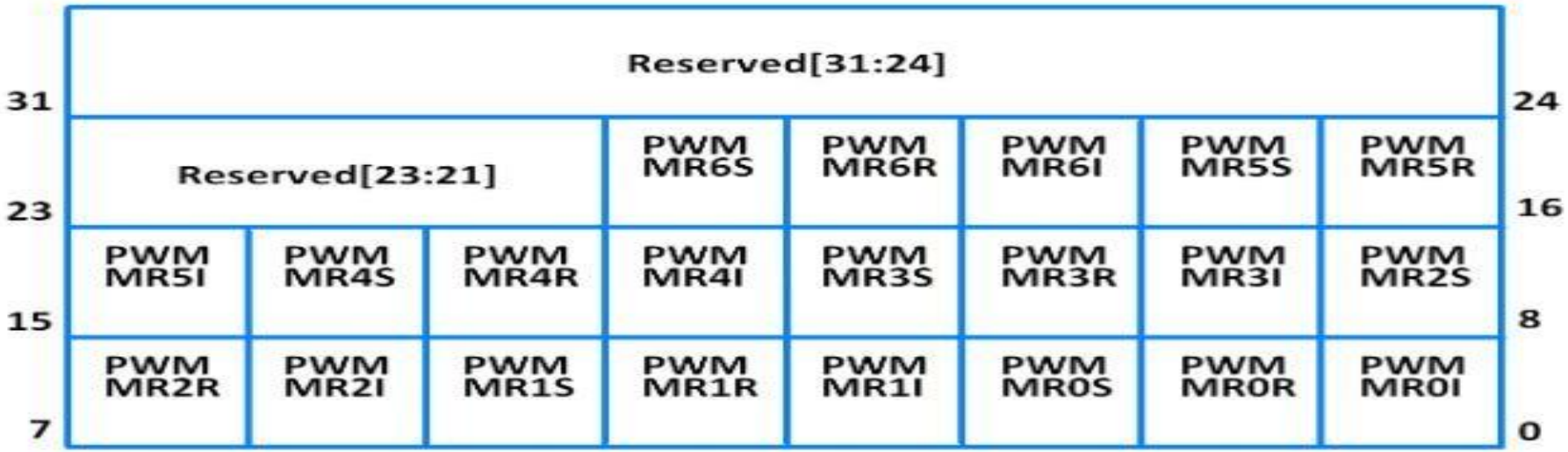
- It is a 32-bit register.
- It controls the division of PCLK by some constant value before it is applied to the PWM Timer Counter.
- It is incremented on every PCLK.
- When it reaches the value in PWM Prescale Register, the PWM Timer Counter is incremented and PWM Prescale Counter is reset on next PCLK.

6. PWMMRo-PWMMR6 (PWM Match Registers)

- These are 32-bit registers.
- The values stored in these registers are continuously compared with the PWM Timer Counter value.
- When the two values are equal, the timer can be reset or stop or an interrupt may be generated.
- The PWMMCR controls what action should be taken on a match.

7. PWMMCR (PWM Match Control Register)

- It is a 32-bit register.
- It controls what action is to be taken on a match between the PWM Match Registers and PWM Timer Counter.



Bit 0 – PWMMR0I (PWM Match register 0 interrupt)

0 = This interrupt is disabled

1 = Interrupt on PWMMR0. An interrupt is generated when PWMMR0 matches the value in PWMTC

Bit 1 – PWMMR0R (PWM Match register 0 reset)

0 = This feature is disabled

1 = Reset on PWMMR0. The PWMTC will be reset if PWMMR0 matches it

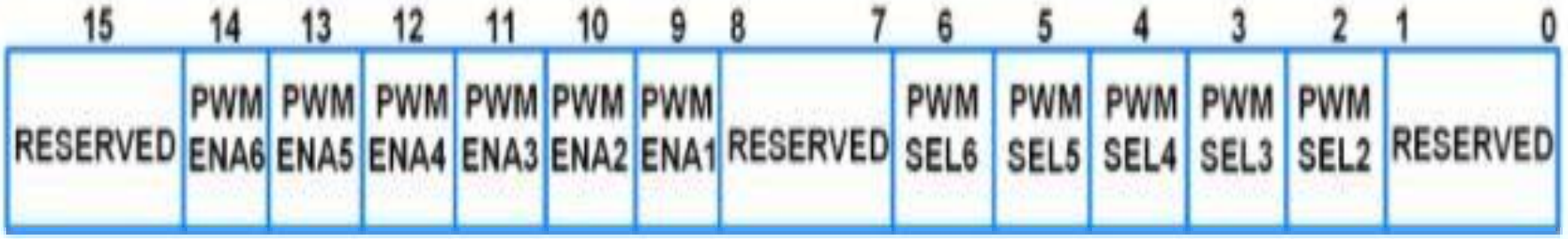
Bit 2 – PWMMR0S (PWM Match register 0 stop)

0 = This feature is disabled

1 = Stop on PWMMR0. The PWMTC and PWMPC is stopped and Counter Enable bit in PWMTCR is set to 0 if PWMMR0 matches PWMTC

PWMMR1, PWMMR2, PWMMR3, PWMMR4, PWMMR5 and PWMMR6 has same function bits (stop, reset, interrupt) as in PWMMR0.

- **Bit 2 – PWMSEL2**
  0 = Single edge controlled mode for PWM2
  1 = Double edge controlled mode for PWM2



- All other PWMSEL bits have similar operation as PWMSEL2 above.
- **Bit 10 – PWMENA2**
  0 = PWM2 output disabled
  1 = PWM2 output enabled
- All other PWMENA bits have similar operation as PWMENA2 above.

9. PWMLER (PWM Latch Enable Register)

- It is an 8-bit register.

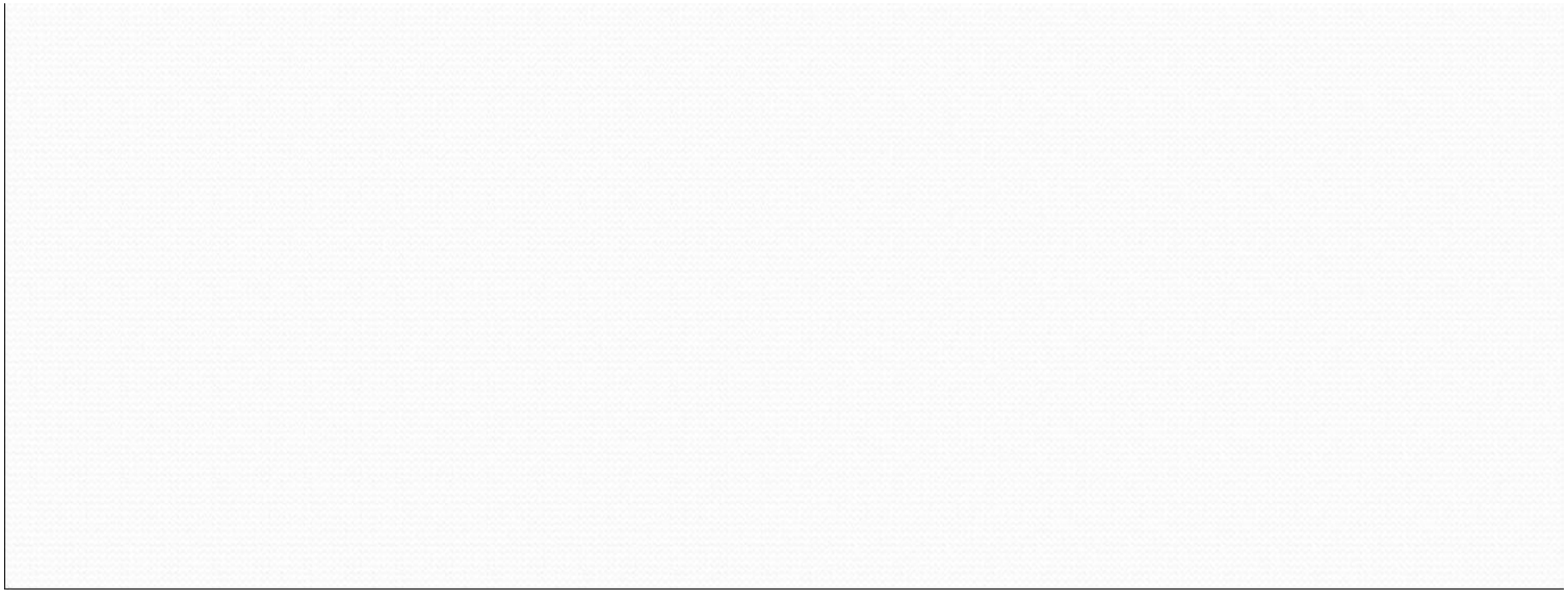
7	6	5	4	3	2	1	0
RESERVED	Enable PWM Match6 Latch	Enable PWM Match5 Latch	Enable PWM Match4 Latch	Enable PWM Match3 Latch	Enable PWM Match2 Latch	Enable PWM Match1 Latch	Enable PWM Match0 Latch

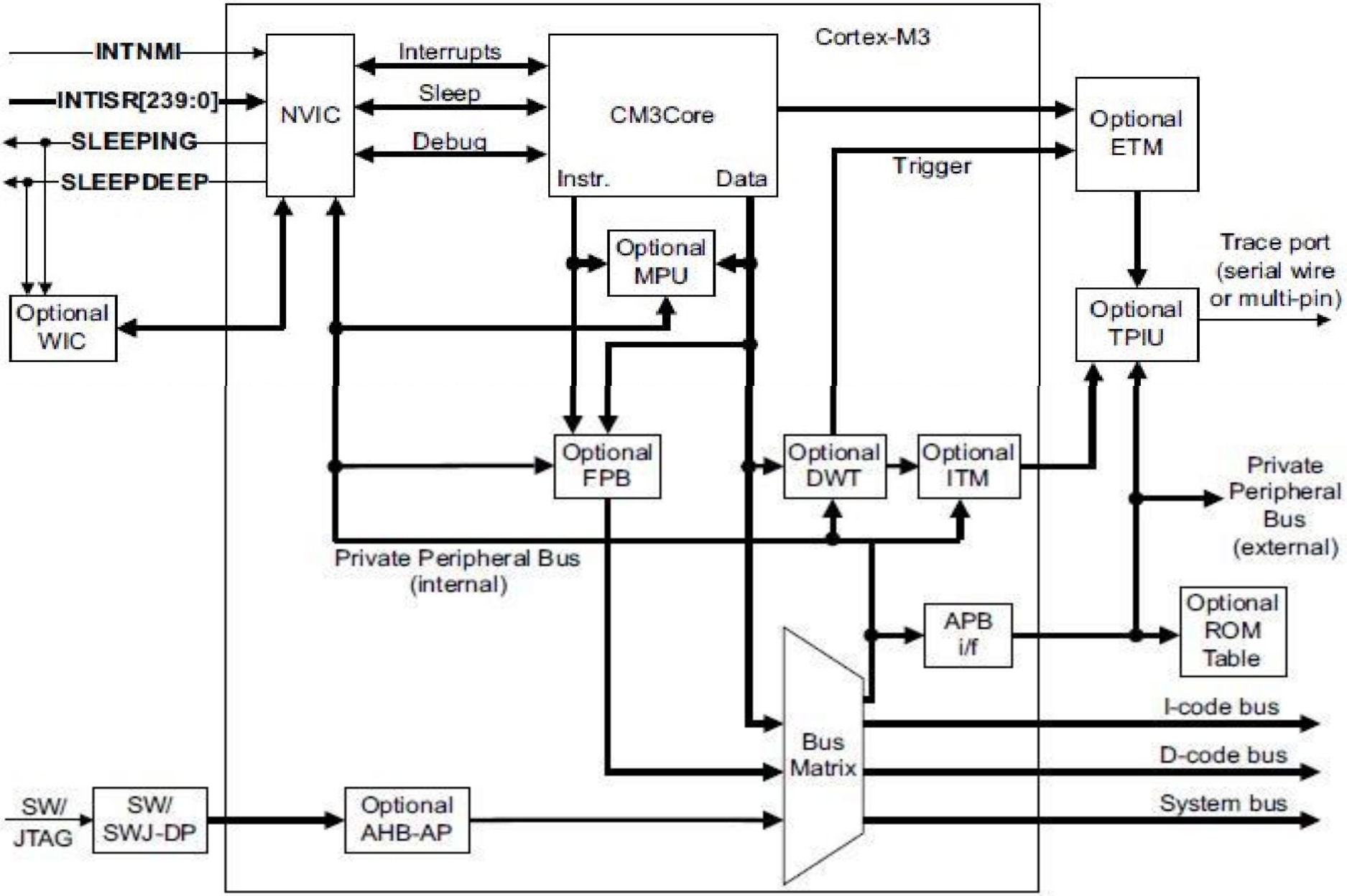
- It is used to control the update of the PWM Match Registers when they are used for PWM generation.
- When a value is written to a PWM Match Register while the timer is in PWM mode, the value is held in the shadow register. The contents of the shadow register are transferred to the PWM Match Register when the timer resets (PWM Match 0 event occurs) and if the corresponding bit in PWMLER is set.
- **Bit 6 – Enable PWM Match 6 Latch**
  Writing a 1 to this bit allows the last written value to PWMMR6 to become effective when timer next is reset by the PWM match event.
- Similar description as that of Bit 6 for the remaining bits.

Steps for PWM generation

- Reset and disable PWM counter using PWMTCR
- Load prescale value according to need of application in the PWMPR
- Load PWMMRo with a value corresponding to the time period of your PWM wave
- Load any one of the remaining six match registers (two of the remaining six match registers for double edge controlled PWM) with the ON duration of the PWM cycle. (PWM will be generated on PWM pin corresponding to the match register you load the value with).
- Load PWMMCR with a value based on the action to be taken in the event of a match between match register and PWM timer counter.
- Enable PWM match latch for the match registers used with the help of PWMLER
- Select the type of PWM wave (single edge or double edge controlled) and which PWMs to be enabled using PWMPCR
- Enable PWM and PWM counter using PWMTCR

Block diagram of ARM CORTEX M3 MCU




Cortex-M3
INTNMI
INTISR[239:0]
SLEEPING
SLEEPDEEP
NVIC
Interrupts
Sleep
Debug
CM3Core
Instr.
Data
Optional ETM
Trigger
Optional MPU
Trace port (serial wire or multi-pin)
Optional WIC
Optional TPIU
Optional FPB
Optional DWT
Optional ITM
Private Peripheral Bus (external)
Private Peripheral Bus (internal)
APB i/f
Optional ROM Table
Bus Matrix
I-code bus
D-code bus
System bus
SW/ JTAG
SW/ SWJ-DP
Optional AHB-AP

- **INTNMI**- Non-maskable interrupt
- **INTISR[239:0]**- External interrupt signals
- **SLEEPING**- Indicates that the Cortex-M3 clock can be stopped.
- **SLEEPDEEP** - Indicates that the Cortex-M3 clock can be stopped
- **WIC** - Wake-up Interrupt Controller
- **NVIC**- Nested Vectored Interrupt Controller
- **ETM**- Embedded Trace Macrocell
- The ETM is an optional debug component that enables reconstruction of program execution. The ETM is designed to be a high-speed, low-power debug tool that only supports instruction trace

- **MPU**- Memory Protection Unit
- The MPU provides full support for:
- protection regions
- overlapping protection regions, with ascending region priority:
- — 7 = highest priority
- — 0 = lowest priority.
- access permissions
- exporting memory attributes to the system.

- **FPB**-Flash Patch and Breakpoint
  - unit to implement breakpoints and code patches.
- ***DWT*** -Data Watchpoint and Trace () unit to implement watchpoints, trigger resources, and system profiling.
- ***ITM****- Instrumentation Trace Macrocell for application-driven trace source that* supports printf style debugging.
- **TPIU**- Trace Port Interface Unit
  - it is an optional component that acts as a bridge between the on-chip trace data from the *Embedded Trace Macrocell (ETM) and the Instrumentation Trace Macrocell*(ITM), with separate IDs, to a data stream, encapsulating IDs where required, that is then captured by a *Trace Port Analyzer (TPA).*

- **SW/SWJ-DP** - SW-DP or SWJ-DP debug port interfaces.
- The debug port provides debug access to all registers and memory in the system, including the processor registers.
- The SW/SWJ-DP might not be present in the production device if no debug functionality is present in the implementation.

UNIT III

EMBEDDED PROGRAMMING

Syllabus

Components for embedded programs- Models of programs- Assembly, linking and loading – compilation techniques- Program level performance analysis – Software performance optimization – Program level energy and power analysis and optimization – Analysis and optimization of program size- Program validation and testing

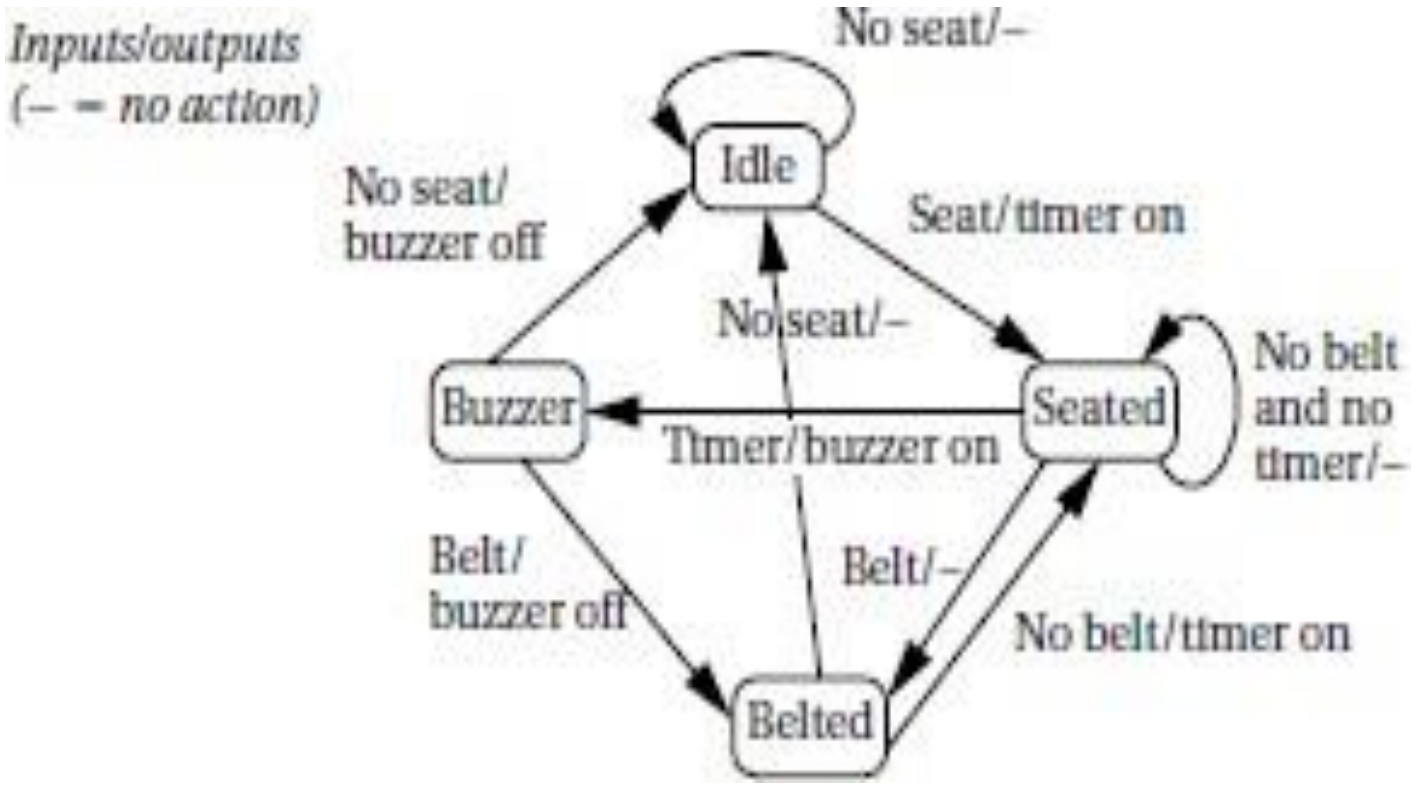
1. COMPONENTS FOR EMBEDDED PROGRAMS

- Embedded components are given by State machine, Circular buffer, and the Queue.

STATE MACHINE

- The reaction of most systems can be characterized in terms of the input received and the current state of the system.
- The finite-state machine style of describing the reactive system's behavior..
- Finite-state machines are usually first encountered in the context of hardware design.

software state machine



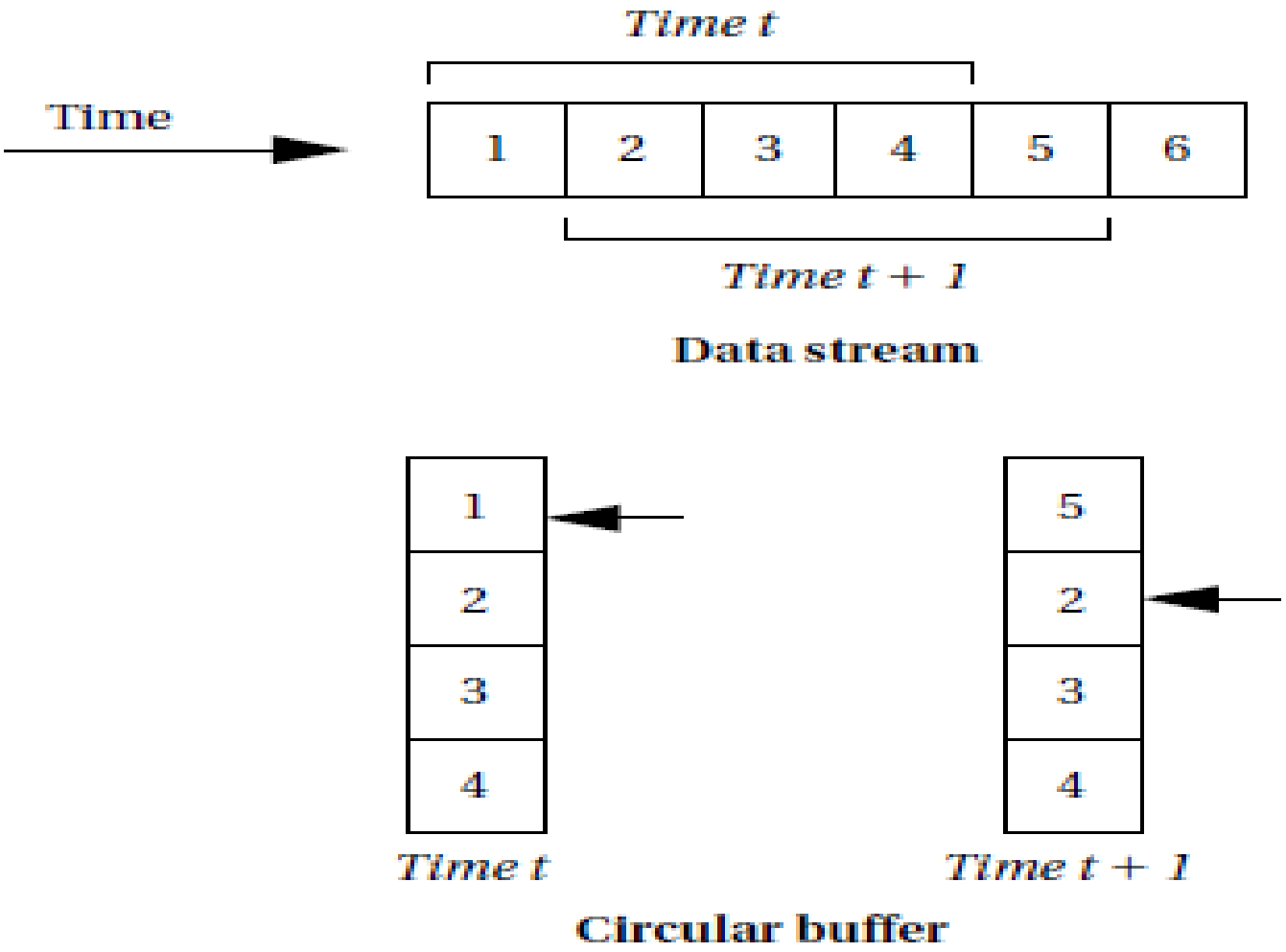
seat, belt, timer

```
#define IDLE 0
#define SEATED 1
#define BELTED 2
#define BUZZER 3
switch (state) { /* check the current state
    */
case IDLE:
if (seat) { state = SEATED;
timer_on = TRUE; }
/* default case is self-loop */
break;
case SEATED:
if (belt) state = BELTED; /* won't hear the
buzzer */
else if (timer) state = BUZZER; /* didn't
    put on
belt in time */
/* default is self-loop */
break;
case BELTED:
if (!seat) state = IDLE; /* person left */
else if (!belt) state = SEATED; /* person
    still
in seat */
break;
case BUZZER:
if (belt) state = BELTED; /* belt is on—
    turn off
buzzer */
else if (!seat) state = IDLE; /* no one in
seat—turn off buzzer */
break;
}
```

Stream-Oriented Programming and Circular Buffers

- The circular buffer is a data structure that handle streaming data in an efficient way.
- Size of the window does not change.
- Fixed-size buffer to hold the current data.
- To avoid constantly copying data within the buffer, move the head of the buffer in time.
- The buffer points to the location at which the next sample will be placed.
- Every time add a sample, automatically overwrite the oldest sample, which is the one that needs to be thrown out.
- When the pointer gets to the end of the buffer, it wraps around to the top.

Circular buffer for streaming data.



1.3 QUEUES

- Queues are also used in signal processing and event processing.
- Queues are used whenever data may arrive and depart at somewhat unpredictable times or when variable amounts of data may arrive.
- A queue is often referred to as an Elastic buffer.

2.MODELS OF PROGRAMS

- Programs are collection of instructions to execute a specified task.
- Models for programs are more general than source code.
- source code can't be used directly because of different type s such as assembly language,C code.
- Single model to describe all of them.
- control/data flow graph (CDFG)→it is the fundamental model for programs

2.1 DATA FLOW GRAPH

- A data flow graph is a model of a program with no conditionals.
- In a high-level programming language, a code segment with no conditionals have only one entry and exit point—is known as a basic block.

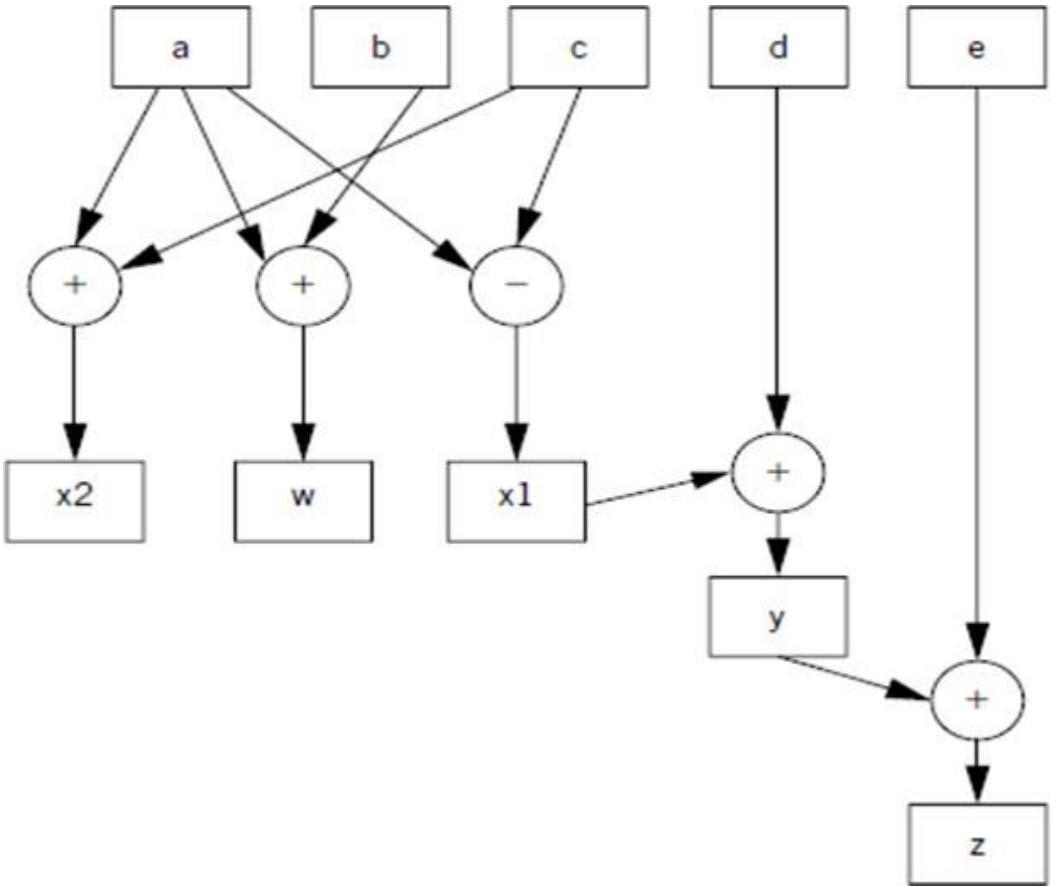
  - A basic block in C

```
w = a + b;
x = a - c;
y = x + d;
x = a + c;
z = y + e;
```

An extended data flow graph for our sample basic block

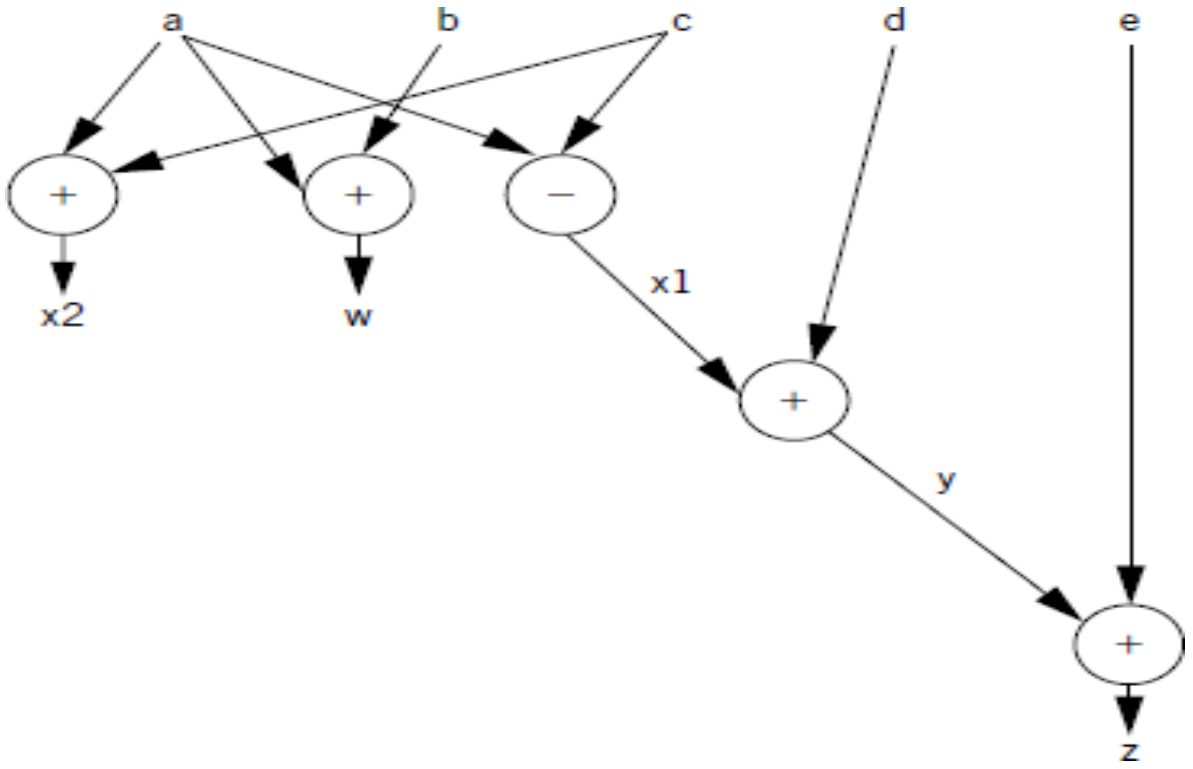
•The basic block in single-assignment form

```
w = a + b;
x1 = a - c;
y = x1 + d;
x2 = a + c;
z = y + e;
```



•Round nodes→denote operators
•Square nodes→denote values.
•The value nodes may be either inputs(a,b) or variables(w,x1).

Standard data flow graph for our sample basic block



2.2. Control/Data Flow Graphs(CDFG)

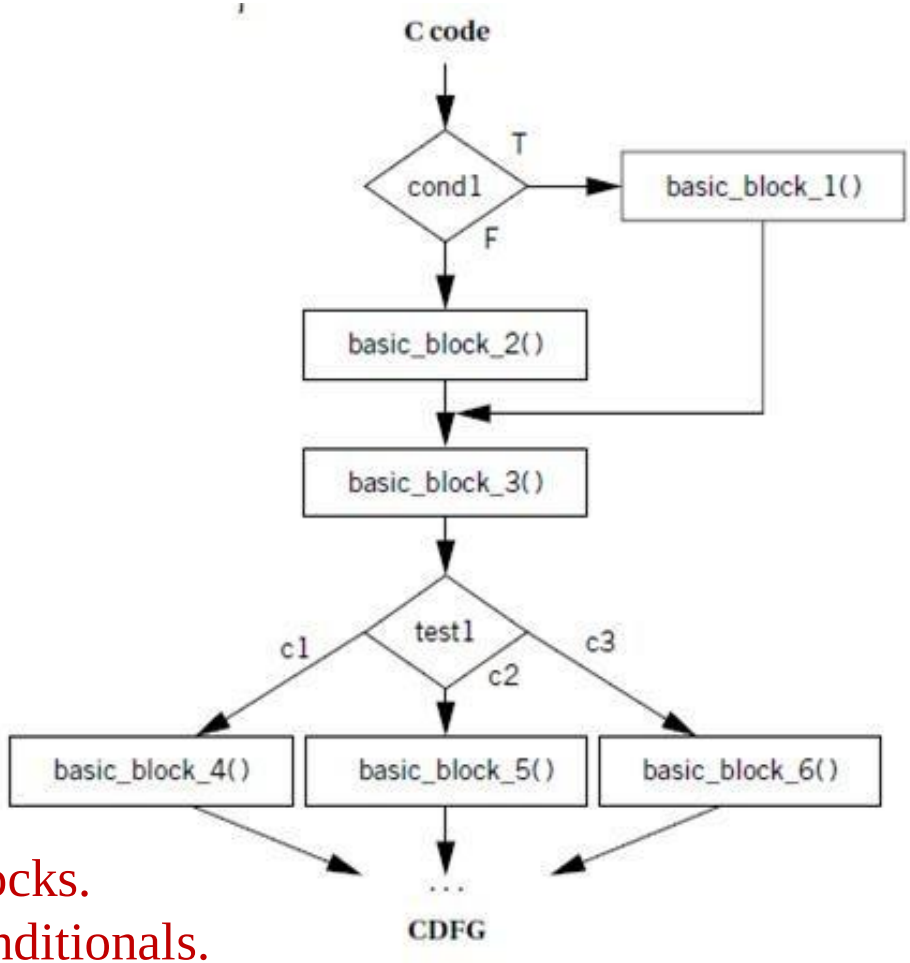
- A CDFG uses a data flow graph as an element,adding constructs to describe control.

CDFG having following two types of nodes.

1. Decision nodes→used to describe the control in a sequential program
- Data flow nodes→ encapsulates a complete data flow graph to represent a data.

C code and

```
if (cond1)
basic_block_1( );
else
basic_block_2();
basic_block_3( );
switch (test1) {
case c1: basic_block_4( ); break;
case c2: basic_block_5( ); break;
case c3: basic_block_6( ): break;
}
```



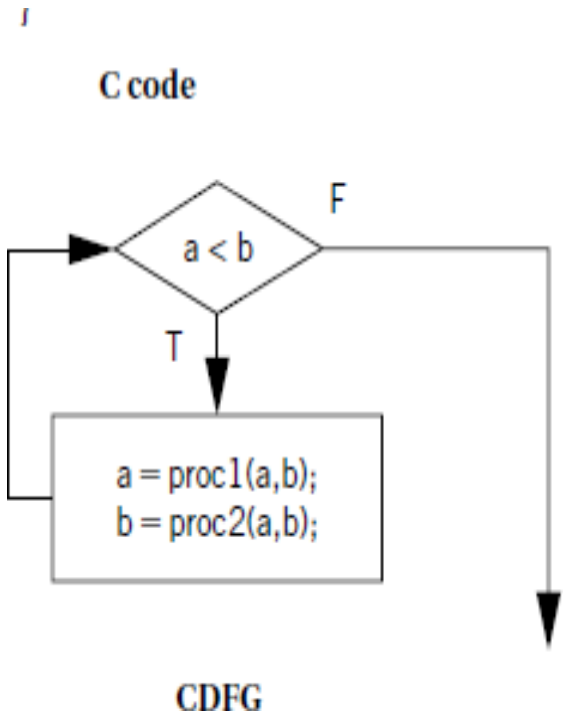
- Rectangular nodes → represent the basic blocks.
- Diamond-shaped nodes → represent the conditionals.
- Label → node's condition
- Edges are labeled with the possible outcomes of evaluating the condition

CDFG for a while loop

```
while (a < b) {
a5proc1(a,b);
b5proc2(a,b);
}
```

CDFG for a while loop

```
while (a < b) {
a5proc1(a,b);
b5proc2(a,b);
}
```

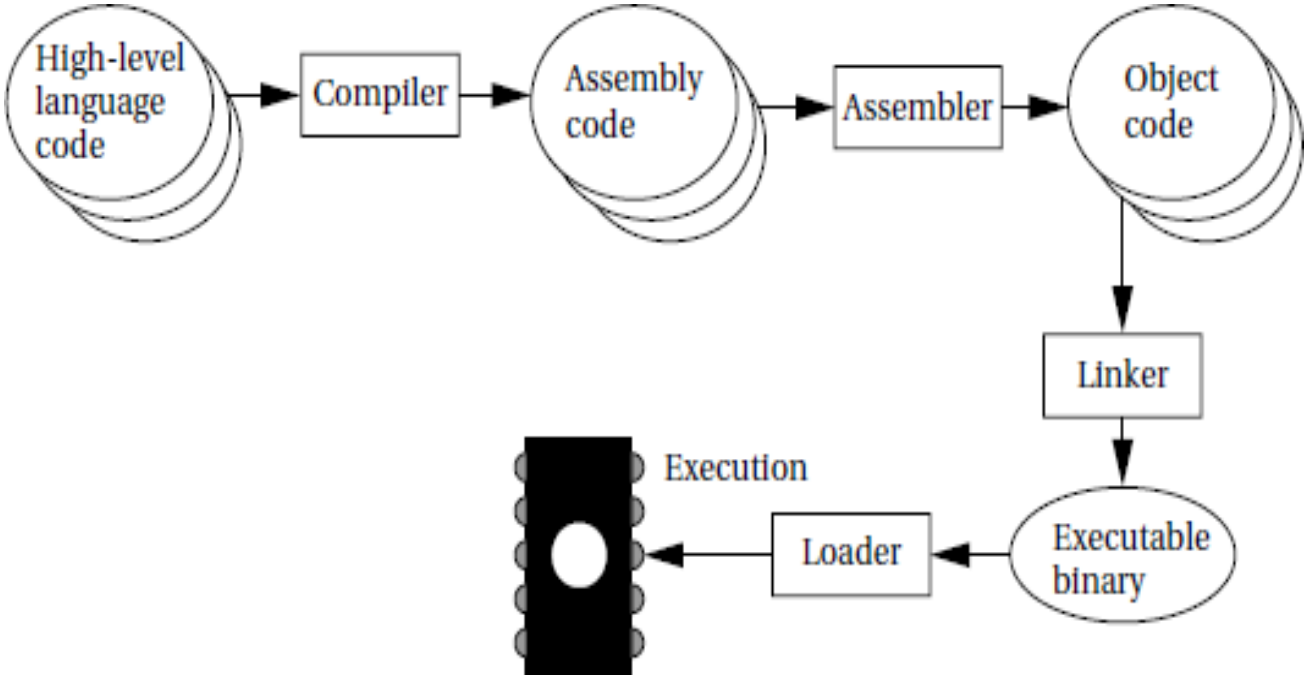


3. ASSEMBLY, LINKING AND LOADING

•Assembly and linking→ last steps in the compilation process

•They convert list of instructions into an image of the program's bits in memory.

•Loading→ puts the program in memory so that it can be executed.



- Compilers → used to create the instruction-level program in to assembly language code.
- Assembler's→ used to translate symbolic assembly language statements into bit-level representations of instructions known as object code and also translating labels into addresses.
- Linker→ determining the addresses of instructions.
- Loader→ load the program into memory for execution.
- Absolute addresses→ Assembler assumes that the starting address of the ALP has been specified by the programmer.
- Relative addresses→ specifying at the start of the file address is to be computed later.

3.1 Assemblers

- Assembler→ Translating assembly code into object code also assembler must translate opcodes and format the bits in each instruction, and translate labels into addresses.
- Labels →it is an abstraction provided by the assembler.
- Labels→know the locations of instructions and data.

Label processing requires making two passes

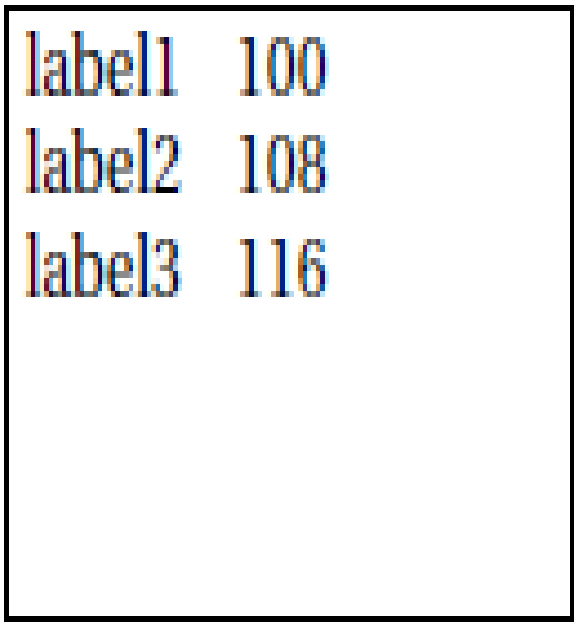
1. first pass scans the code to determine the address of each label.
2. second pass assembles the instructions using the label values computed in the first pass.

EXAMPLE

CODE

```
        ORG 100
label1  ADR r4,c
        LDR r0,[r4]
label2  ADR r4,d
        LDR r1,[r4]
label3  SUB r0,r0,r1
```

SYMBOL TABLE



Symbol table

3.2)LINKING

- A linker allows a program to be stitched together out of several smaller pieces.
- The linker operates on the object files and links between files.
- Some labels will be both defined and used in the same file.
- Other labels will be defined in a single file but used elsewhere .
- The place in the file where a label is defined is known as an entry point.
- The place in the file where the label is used is called an external reference.

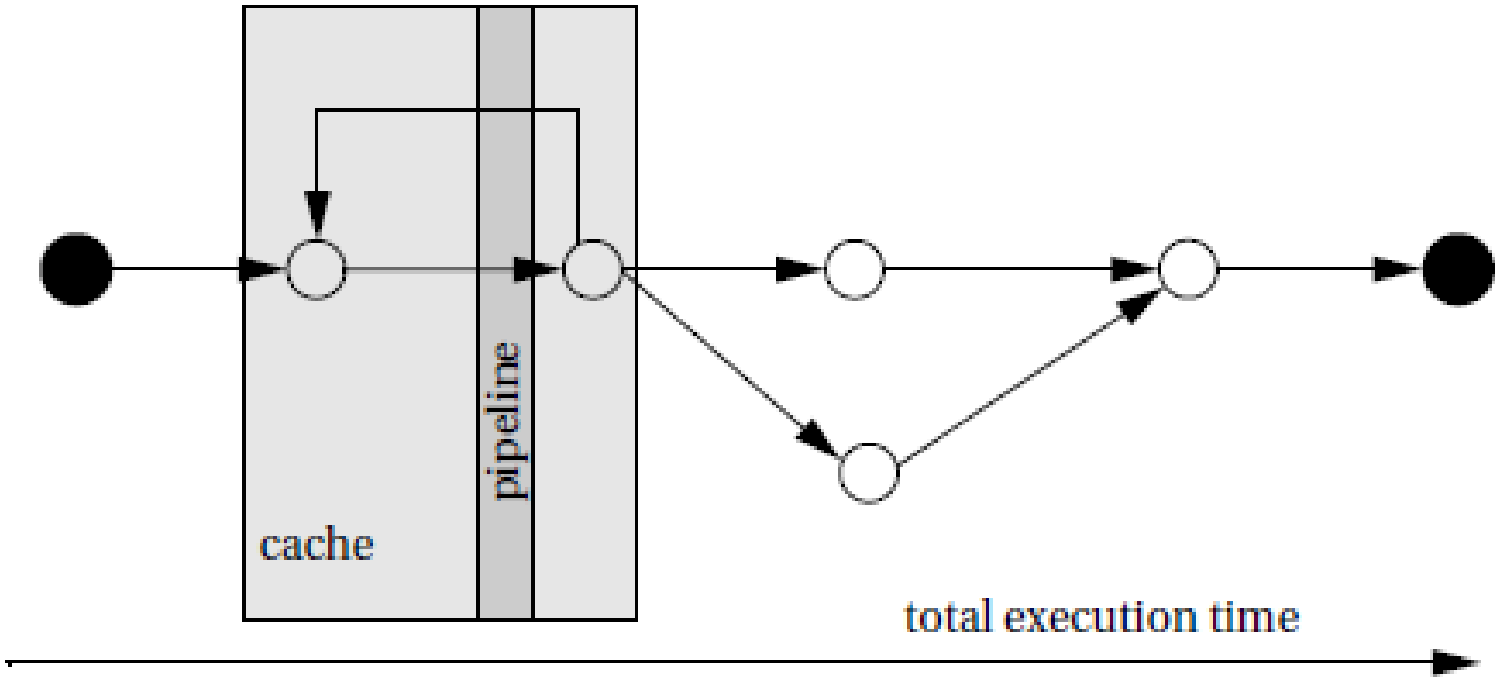
Phases of linker

- First Phase→it determines the address of the start of each object file
- Second Phase→the loader merges all symbol tables from the object files into a single,large table.

4. PROGRAM-LEVEL PERFORMANCE ANALYSIS

- The techniques we use to analyze program execution time are also helpful in analyzing properties such as power consumption.
- The CPU executes the entire program at the rate we desire.
- The execution time of a program often varies with the input data values.
- The cache has a major effect on program performance.
- Cache's behavior depends in part on the data values input to the program.
- The execution time of an instruction in a pipeline depends not only on that instruction but on the instructions around it in the pipeline.

Execution time of a program



Program Performance Measuring techniques

1. Simulator

- It runs on a PC, takes as input an executable for the microprocessor along with input data, and simulates the program.

2. Timer

- It is can be used to measure performance of executing sections of code.
- The length of the program that can be measured is limited by the accuracy of the timer.

3. Logic analyzer

- It is used to measure the start and stop times of a code segment.
- The length of code that can be measured is limited by the size of the logic analyzer's buffer.

4.2)Types of performance Parameters

1. Average-case execution time

- This is the typical execution time we would expect for typical data.

2. Worst-case execution time

- The longest time that the program can spend on any input sequence is clearly important for systems that must meet deadlines.

3. Best-case execution time

- This measure can be important in multi-rate real-time systems.

4.3) Elements of Program Performance

- Execution time =Program path +Instruction timing
- Program path→ It is the sequence of instructions executed by the program.
- Instruction timing→It is determined based on the sequence of instructions traced by the program path.
- Not all instructions take the same amount of time.
- The execution time of an instruction may depend on operand values.

Measurement-Driven Performance Analysis

- To measure the program's performance →need CPU or its simulator .
- Measuring program performance → combination of determination of the execution path and the timing of that path.
- program trace → record of the execution path of a program.

Cycle-Accurate Simulator

- It can determine the exact number of clock cycles required for execution.
- It is built with detailed knowledge of how the processor works .
- It is slower than the processor itself, but a variety of techniques can be used to make them surprisingly fast.
- It has a complete model of the processor, including the cache.
- It can provide information about why the program runs too slowly.

5. SOFTWARE PERFORMANCE OPTIMIZATION

5.1)Loop Optimizations-Loops are important targets for optimization because programs with loops tend to spend a lot of time executing those loops.

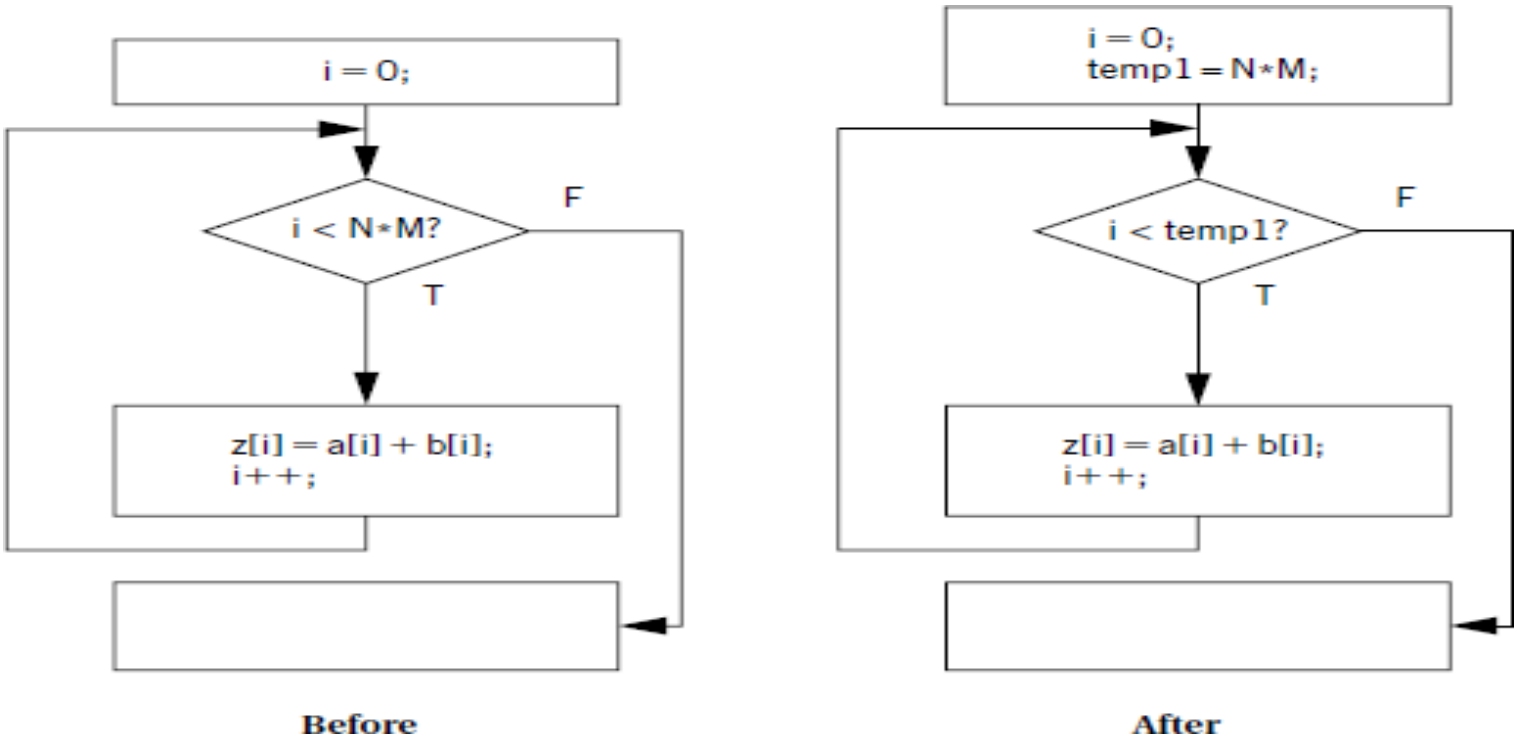
- Code motion
- Induction variable elimination
- Strength reduction

Code motion

- It can move unnecessary code out of a loop.
- If a computation's result does not depend on operations performed in the loop body,thenwe can safely move it out of the loop.

```
for (i = 0; i < N*M; i++)
 {
z[i] = a[i] + b[i];
}
```

Code motion in a loop



•The loop bound computation is performed on every iteration during the loop test, even though the result never changes.

•We can avoid *N X M- 1 unnecessary executions of this statement by moving it before the loop.*

Induction variable elimination

- It is a variable whose value is derived from the loop iteration variable's value.
- The compiler often introduces induction variables to help it implement the loop.
- Properly transformed→ able to eliminate some variables and apply strength reduction to others.
- A nested loop is a good example of the use of induction variables.

```
for (i = 0; i < N; i++)
for (j = 0; j < M; j++)
z[i][j] = b[i][j];
```

- The compiler uses induction variables to help it address the arrays. Let us rewrite the loop in C using induction variables and pointers

```
for (i = 0; i < N; i++)
for (j = 0; j < M; j++) {
zbinduct = i*M + j;
*(zptr + zbinduct) = *(bptr + zbinduct);
}
```

Strength reduction

- It reduce the cost of a loop iteration.

Consider the following assignment

```
y = x * 2;
```

- In integer arithmetic, we can use a left shift rather than a multiplication by 2
- If the shift is faster than the multiply, then perform the substitution.
- This optimization can often be used with induction variables because loops are often indexed with simple expressions.

5.2 Cache Optimizations

- A loop nest is a set of loops, one inside the other.
- Loop nests occur when we process arrays.
- A large body of techniques has been developed for optimizing loop nests.
- Rewriting a loop nest changes the order in which array elements are accessed.
- This can expose new parallelism opportunities that can be exploited by later stages of the compiler, and it can also improve cache performance.

6. PROGRAM-LEVEL ENERGY AND POWER ANALYSIS AND OPTIMIZATION

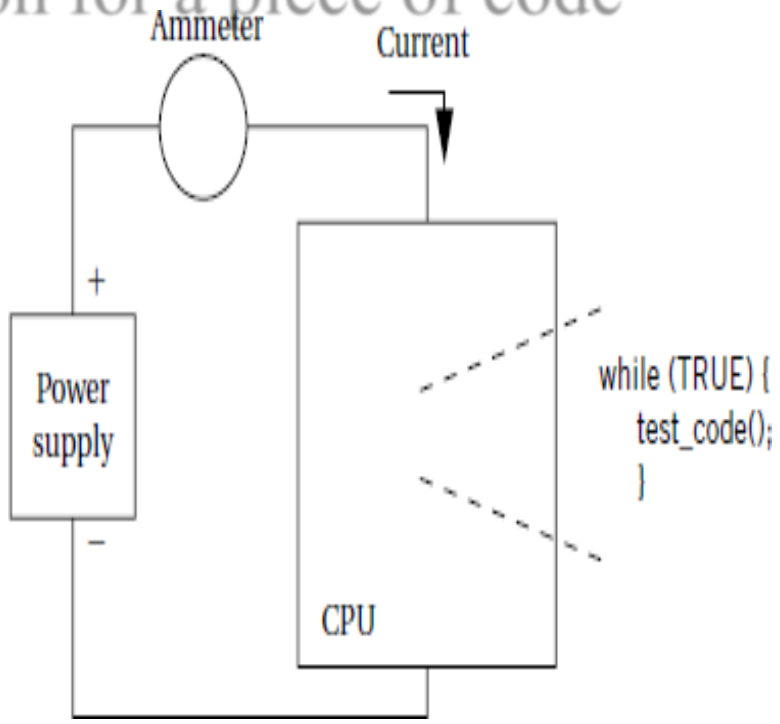
- Power consumption is a important design metric for battery-powered systems.
- It is increasingly important in systems that run off the power grid.
- Fast chips run hot, and controlling power consumption is an important element of increasing reliability and reducing system cost.

Power consumption reduction techniques.

- To replace the algorithms with others that consume less power.
- By optimizing memory accesses ,able to significantly reduce power.
- To turn off the subsystems of CPU, chips in the system, in order to save power.

Measuring energy consumption for a piece of code

- Program's energy consumption→ how much energy the program consumes.
- To measure power consumption for an instruction or a small code fragment.
- It is used to executes the code under test over and over in a loop.
- By measuring the current flowing into the CPU,we are measuring the power consumption of the complete loop, including both the body and other code.
- By separately measuring the power consumption of a loop with no body.
- we can calculate the power consumption of the loop body code as the difference b/w the full loop and the bare loop energy cost of an instruction.



List of the factors contribution for energy consumption of the program.

- Energy consumption varies somewhat from instruction to instruction.
- The sequence of instructions has some influence.
- The opcode and the locations of the operands also matter.

Steps to Improve Energy Consumption

- Try to use registers efficiently(r4)
- Analyze cache behavior to find major cache conflicts.
- Make use of page mode accesses in the memory system whenever possible.
- Moderate loop unrolling eliminates some loop control overhead. when the loop is unrolled too much, power increases.
- Software pipelining reducing the average energy per instruction.
- Eliminating recursive procedure calls where possible saves power by getting rid of function call overhead.
- Tail recursion can often be eliminated, some compilers do this automatically.

7. ANALYSIS AND OPTIMIZATION OF PROGRAM SIZE

- Memory size of a program is determined by the size of its data and instructions.
- Both must be considered to minimize program size.
- Data provide an opportunity to minimizing the size of program.
- Data buffers can be reused at several different points in program, which reduces program size.
- Some times inefficient programs keep several copies of data, identifying and eliminating duplications can lead to significant memory savings.
- Minimizing the size of the instruction text and reducing the number of instructions in a program→ which reduces program size
- Proper instruction selection may reduce code size.
- Special compilation modes produce the program in terms of the dense instruction set.
- Program size of course varies with the type of program, but programs using the dense instruction set are often 70 to 80% of the size of the standard instruction set equivalents.

8. PROGRAM VALIDATION AND TESTING

- Complex systems→ need testing to ensure the working behavior of the systems.
- Software Testing→used to generate a comprehensive set of tests to ensure that our system works properly.
- The testing problem is divided into sub-problems and analyze each sub problem.

Types of testing strategies

1. White/Clear-box Testing→ generate tests ,based on the program structure.
2. Black-box Testing→ generate tests ,without looking at the internal structure of the program.

Clear box testing

- Testing→requires the control/data flow graph of a program's source code.
- To test the program→ exercise both its control and data operations.
- To execute and evaluate the tests→ control the variables in the program and observe the results .

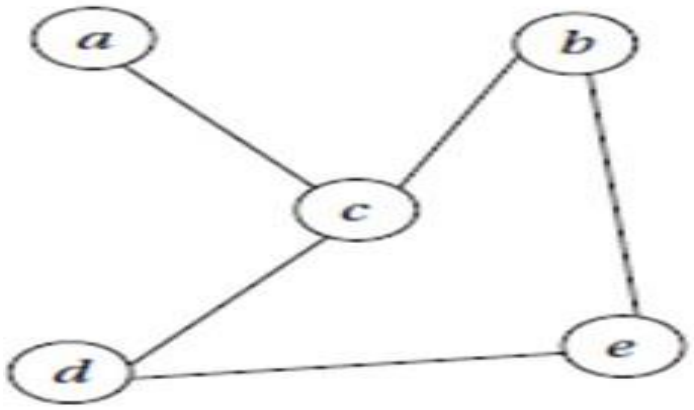
The following three things to be followed during a test

1. Provide the program with inputs for the test.
2. Execute the program to perform the test.
3. Examine the outputs to determine whether the test was successful.

- Execution Path→To test the program by forcing the program to execute along chosen paths. (giving it inputs that it to take the appropriate branches)

Graph Theory

•It help us get a quantitative handle on the different paths required.
•Undirected graph-→form any path through the graph from combinations of basis paths.
•Incidence matrix contains each row and column represents a node.
•1 is entered for each node pair connected by an edge.



Graph

	a	b	c	d	e
a	0	0	1	0	0
b	0	0	1	0	1
c	1	1	0	1	0
d	0	0	1	0	1
e	0	1	0	1	0

Incidence matrix

a	1	0	0	0	0
b	0	1	0	0	0
c	0	0	1	0	0
d	0	0	0	1	0
e	0	0	0	0	1

Basis set

Cyclomatic Complexity

- It is a software metric tool.
- Used to measure the control complexity of a program.

$$M = e - n + 2p.$$

- *e* → *number of edges in the flow graph*
- *n* → *number of nodes in the flow graph*
- *p* → *number of components in the graph*

Types of Clear Box test strategy

1. Branch testing
2. Domain testing
3. Data flow testing

Branch testing

- This strategy requires the true and false branches of a conditional.
- Every simple condition in the conditional's expression to be tested at least once.

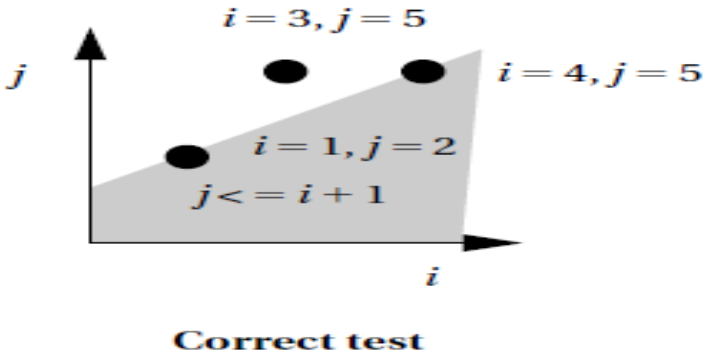
```
if ((x == good_pointer) && (x->field1 == 3))
{ printf("got the value\n"); }
```

The bad code we actually wrote

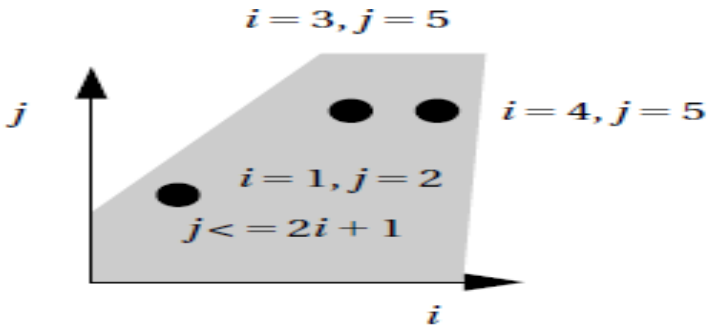
```
if ((x = good_pointer) && (x->field1 == 3))
{ printf("got the value\n"); }
```

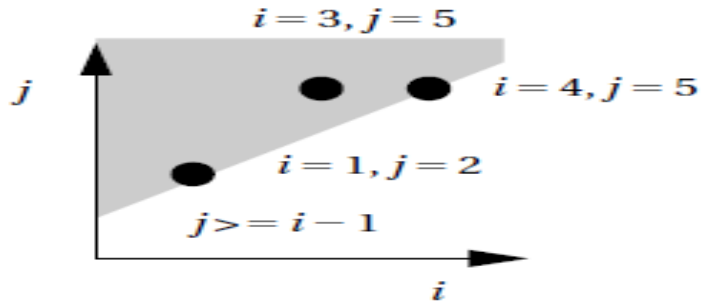
Domain testing

- It concentrates on linear-inequalities.
- The program should use for the test is $j <= i + 1$
- We test the inequality with three test points
- Two on the boundary of the valid region
- Third outside the region but between the *i values of the other* two points.



Correct test

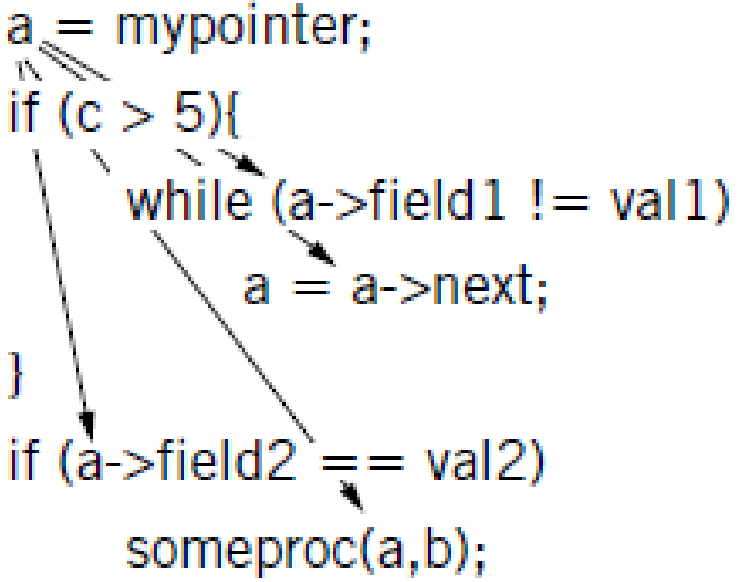




Incorrect tests

Data flow testing

- It use of def-use analysis (definition-use analysis).
- It selects paths that have some relationship to the program's function.
- Compilers→ which use def-use analysis for Optimization.
- A variable's value is defined when an assignment is made to the variable.
- It is used when it appears on the right side of an assignment.



8.2)Block Box Testing

- Black-box tests are generated without knowledge of the code being tested.
- It have a low probability of finding all the bugs in a program.
- We can't test every possible input combination, but some rules help us select reasonable sets of inputs.

1. Random Tests

- Random values are generated with a given inputs.
- The expected values are computed first, and then the test inputs are applied.

2. Regression Tests

- When tests are created during earlier or previous versions of the system.
- Those tests should be saved→ apply to the later versions of the system.
- It simply exercise current version of the code and possibly exercise different bugs.
- In digital signal processing systems→ Signal processing algorithms are implemented to save hardware costs.
- Data sets can be generated for the numerical accuracy of the system.
- These tests can often be generated from the original formulas without reference to the source code.

UNIT IV
REAL TIME SYSTEMS

Structure of a Real Time System - Estimating program run times – Task Assignment and Scheduling – Fault Tolerance Techniques – Reliability, Evaluation – Clock Synchronization.

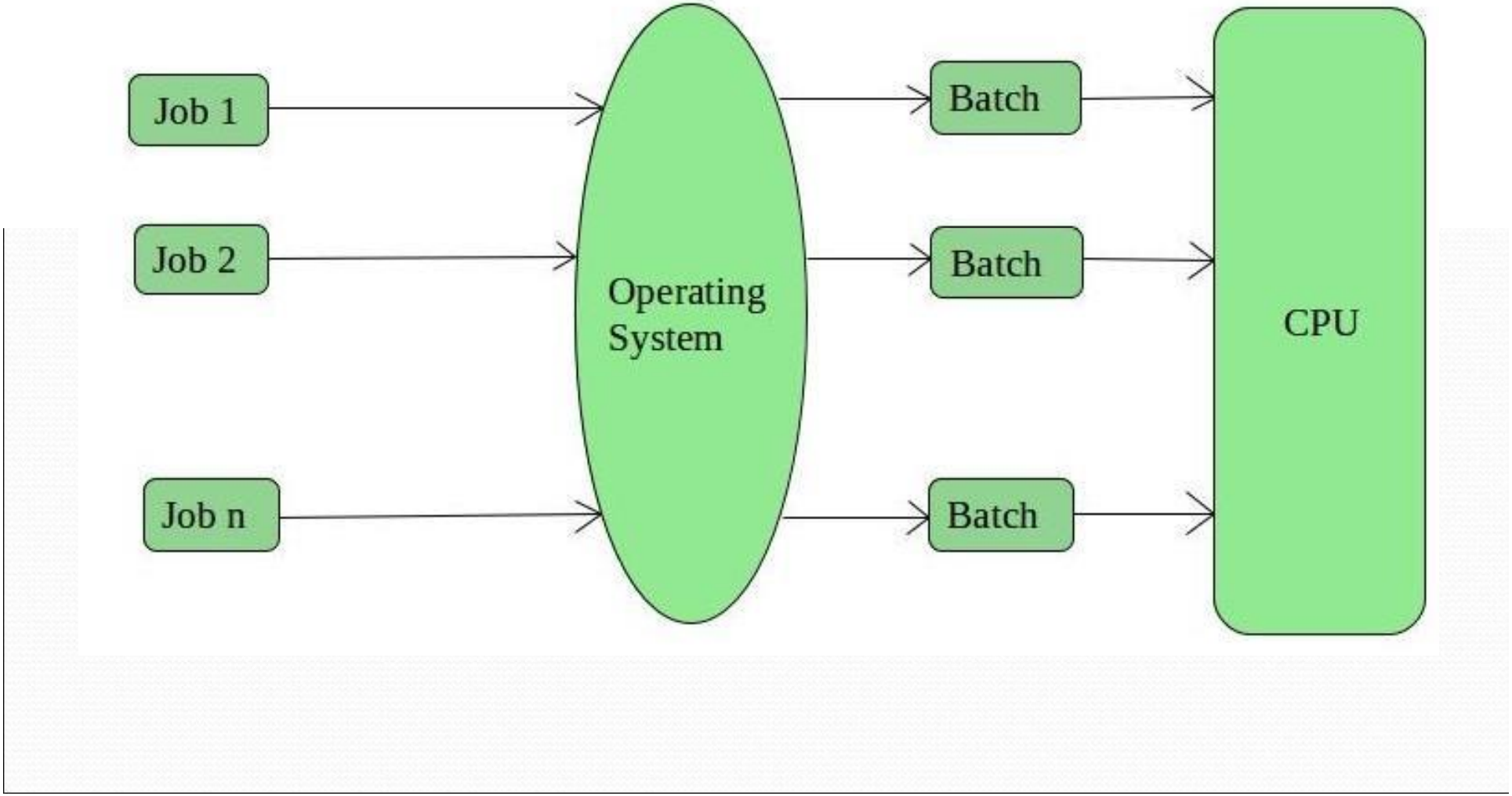
Operating System

- An Operating System performs all the basic tasks like managing file, process, and memory.
- Thus operating system acts as manager of all the resources, i.e. resource manager.
- Thus operating system becomes an interface between user and machine.

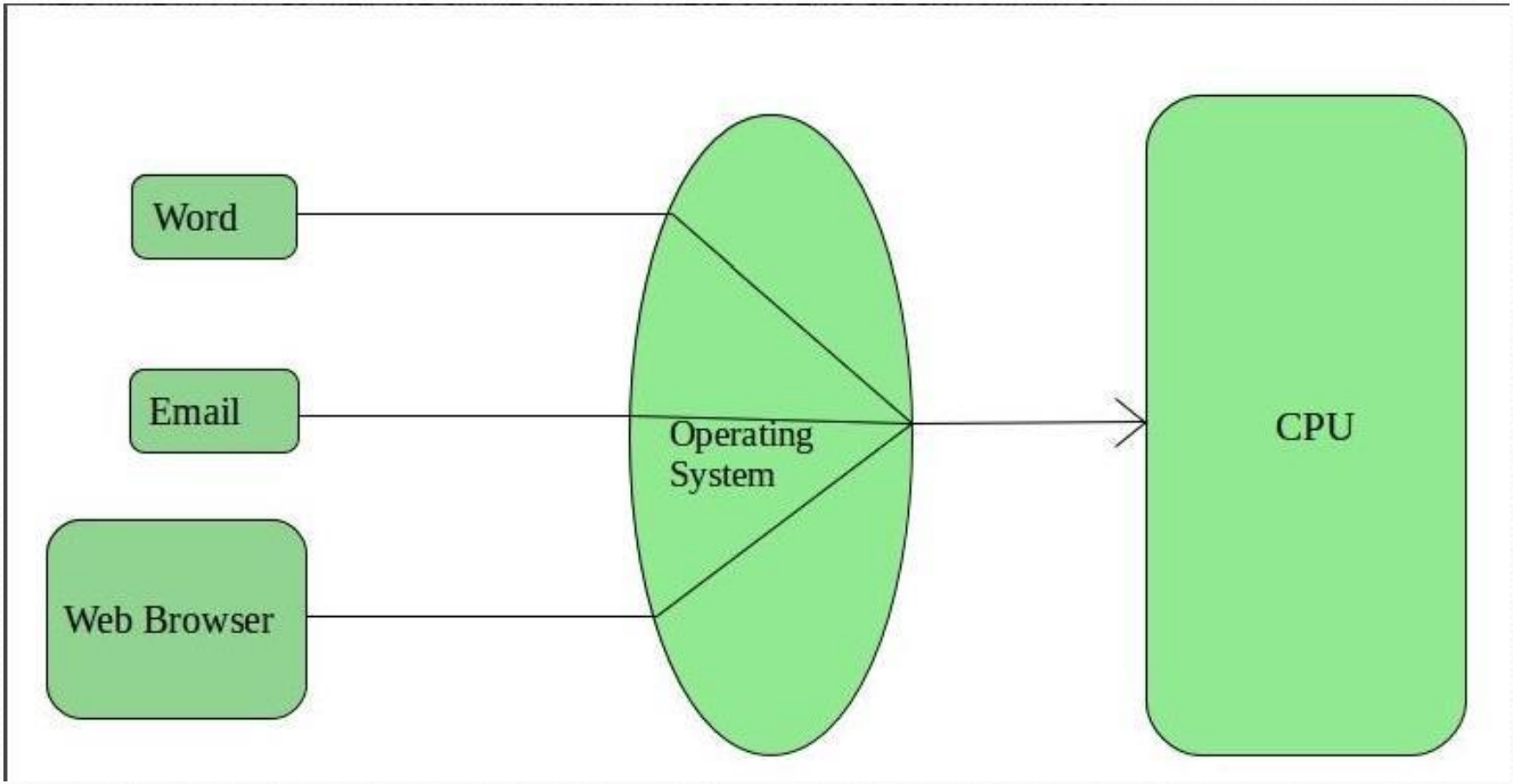
Types of Operating Systems

- **Batch Operating System**
- **Time-Sharing Operating Systems**
- **Distributed Operating System**
- **Network Operating System**
- **Real-Time Operating System**

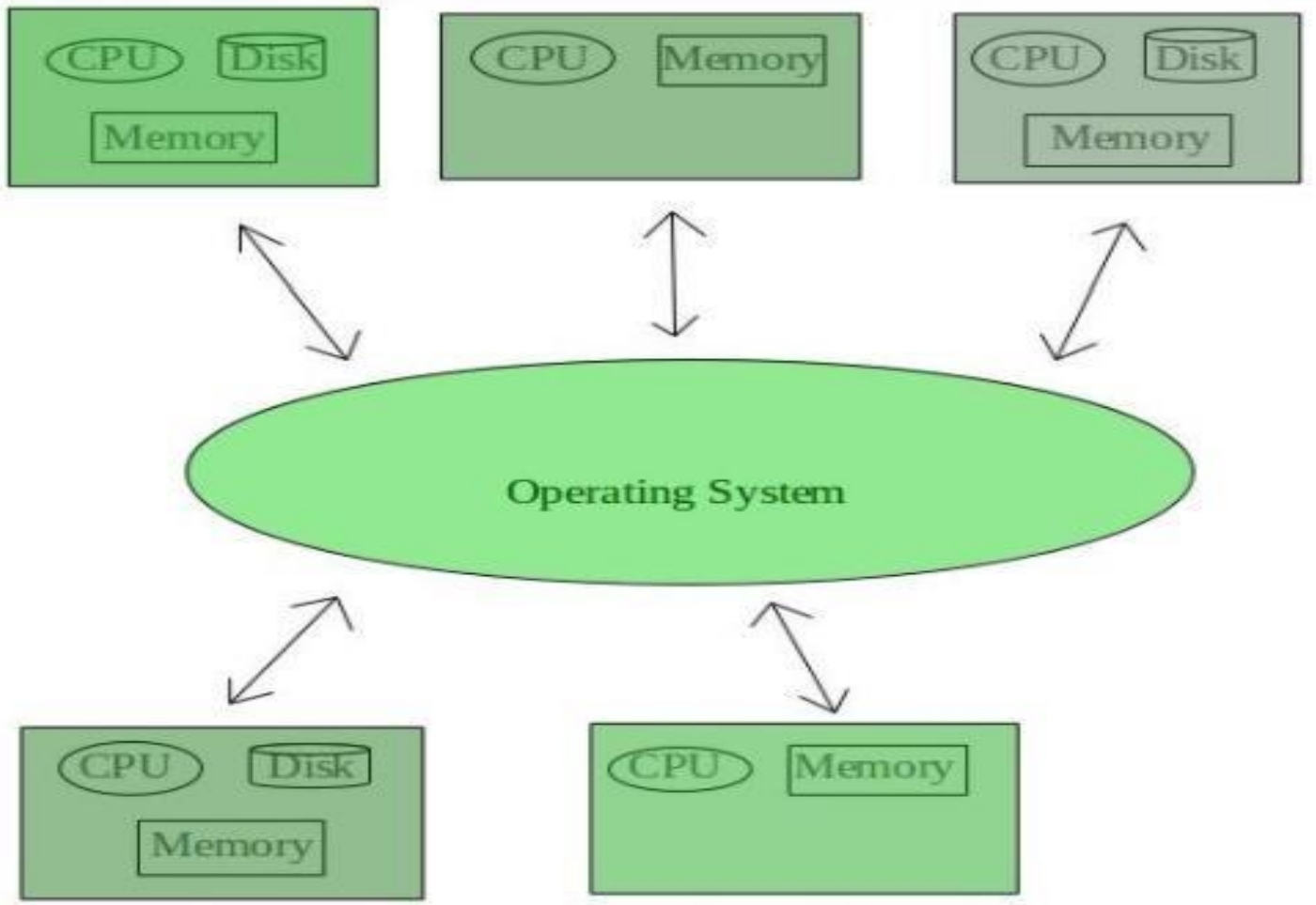
1. Batch Operating System



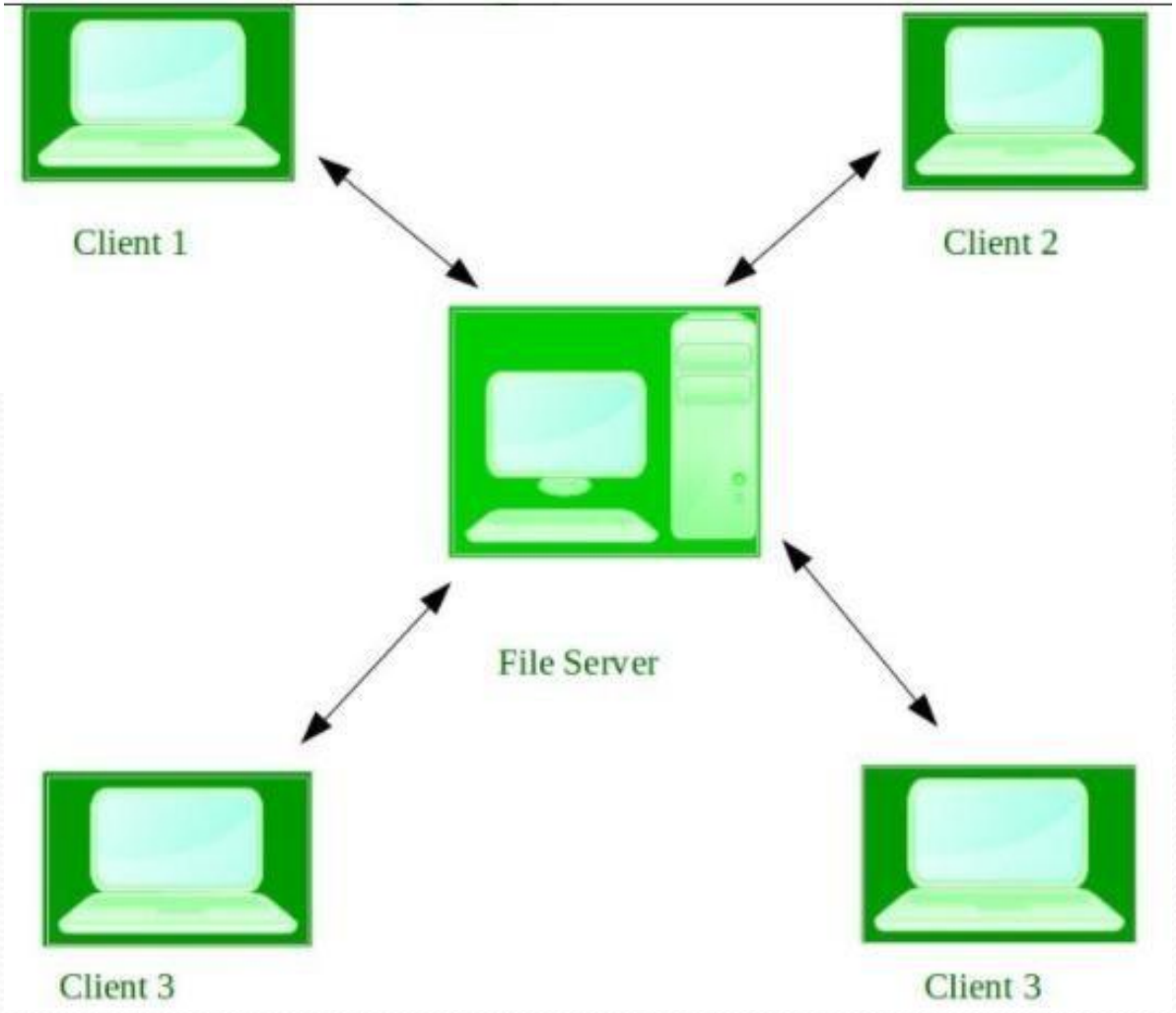
Time-Sharing Operating Systems



Distributed Operating System



Network Operating System



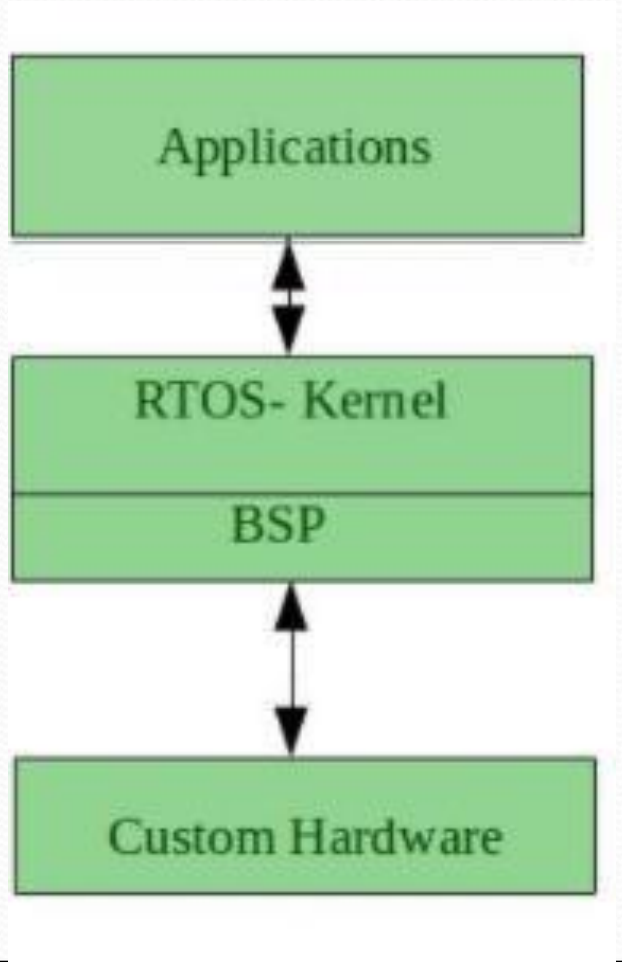
Real-Time Operating System

- These types of OSs serves the real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called **response time**.

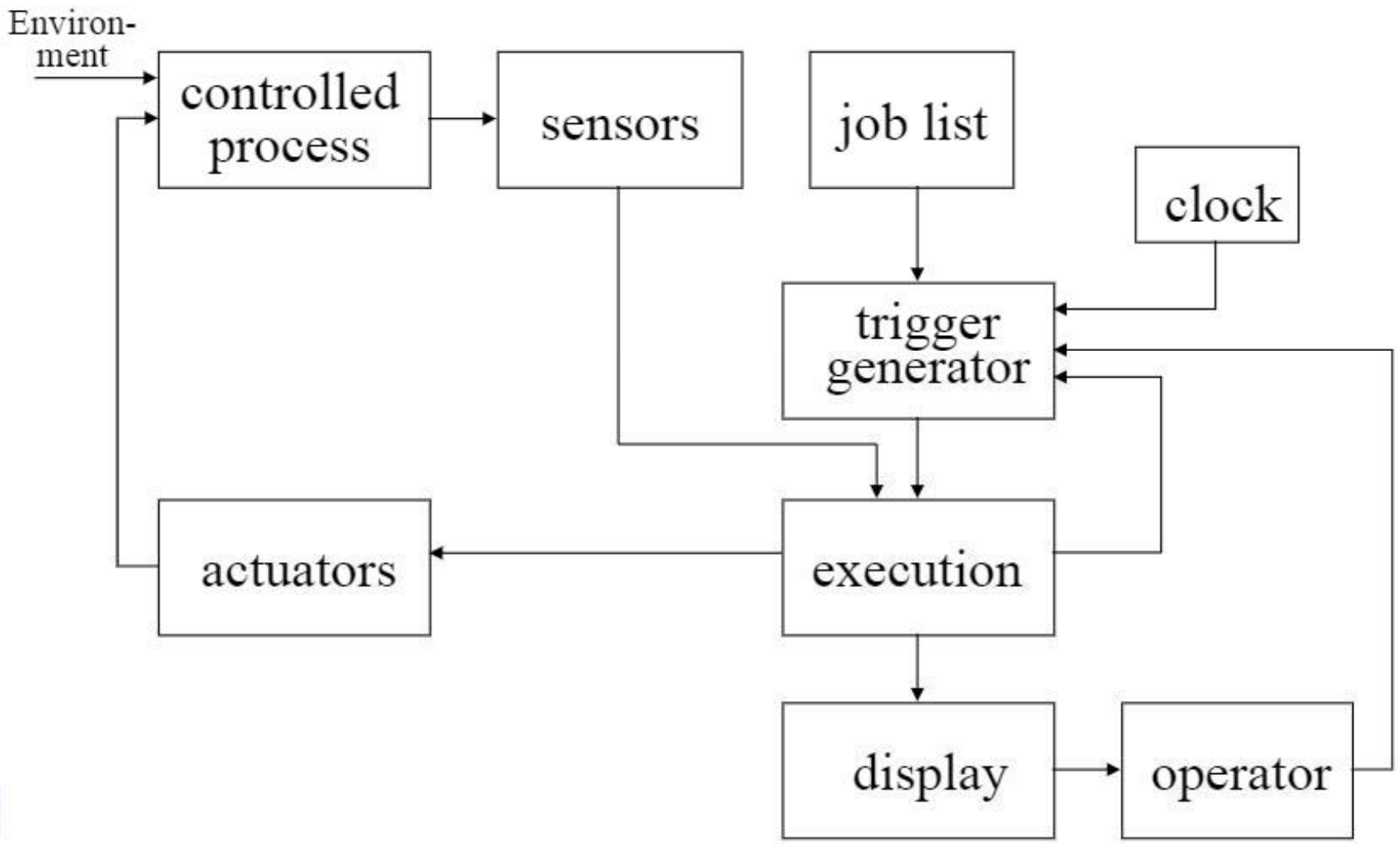
- **Real-time systems** are used when there are time requirements are very strict like missile systems, air traffic control systems, robots etc
- **Two types of Real-Time Operating System which are as follows:**
  - **Hard Real-Time Systems:**
  - **Soft Real-Time Systems:**

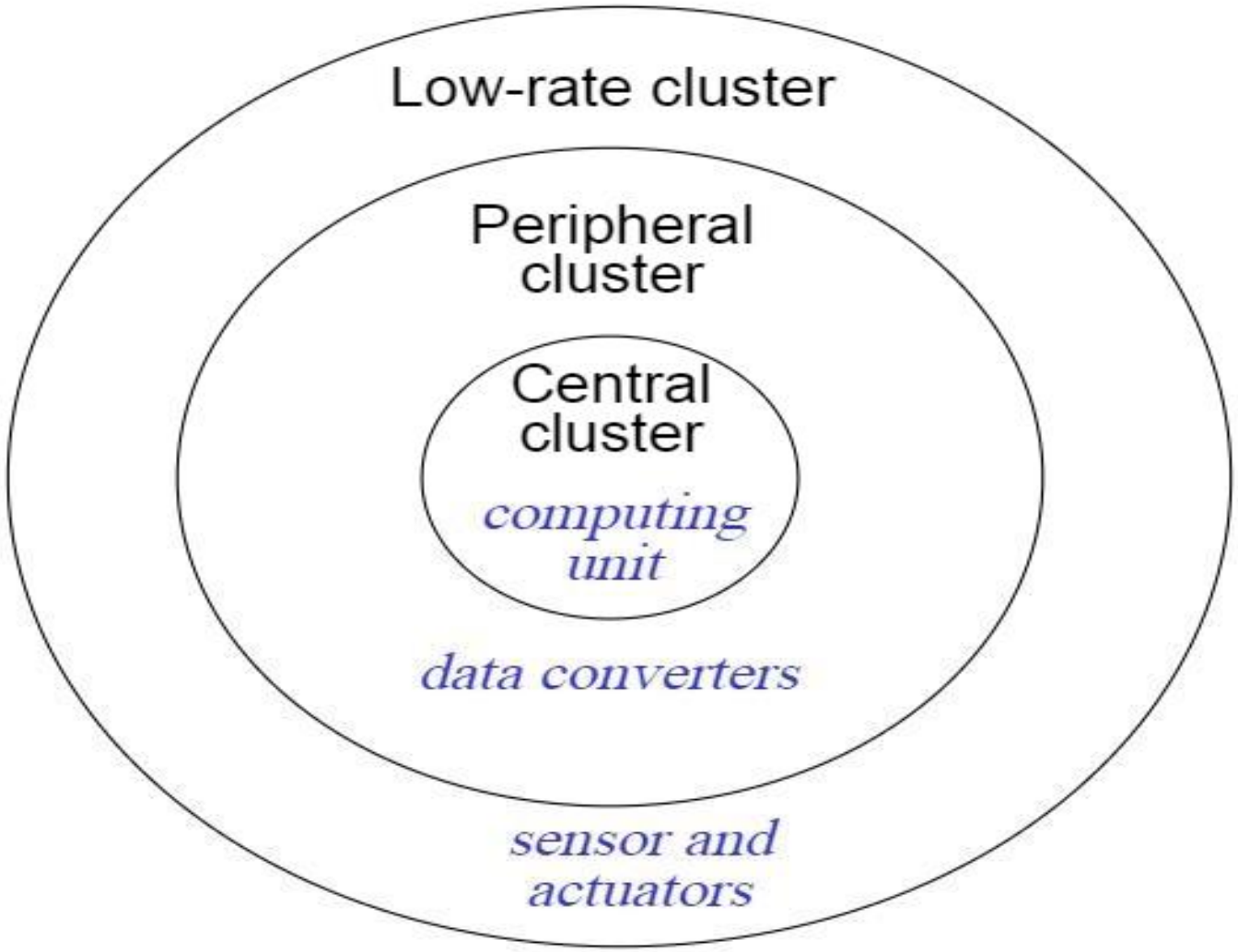
- **Hard Real-Time Systems:**
  These OSs are meant for the applications where time constraints are very strict and even the shortest possible delay is not acceptable.
- These systems are built for saving life like automatic parachutes or air bags which are required to be readily available in case of any accident.
- Virtual memory is almost never found in these systems.

- **Soft Real-Time Systems:**
  These OSs are for applications where for time-constraint is less strict.



Structure of a Real Time System


Environ-
ment
controlled process
sensors
job list
clock
trigger generator
actuators
execution
display
operator


Low-rate cluster
Peripheral cluster
Central cluster
computing unit
data converters
sensor and actuators

4.2 Estimating Program Run Times

- Real time system meet deadlines, it is important to be able to accurately estimate program run times.
- Estimating the executing time of any given program is a very difficult task
- It depend on the following factors
  - Source code
  - Compiler-Mapping should be depend on the compiler used.
  - Machine architecture
  - Operating system

- Analysis of a source code
  - L1: a = b x c;
  - L2: b = d + e;
  - L3: d = a – f;

$$\sum_{}^{3} T_{exec}(L_i)$$

L1.1	Get the address of *c*
L1.2	Load *c*
L1.3	get the address of *b*
L1.4	Load *b*
L1.5	Multiply
L1.6	Store into *a*

- Example 2

```
L4. While (p) do
L5.          Q1;
L6.          Q2;
L7.          Q3;
L8.       End While;
```

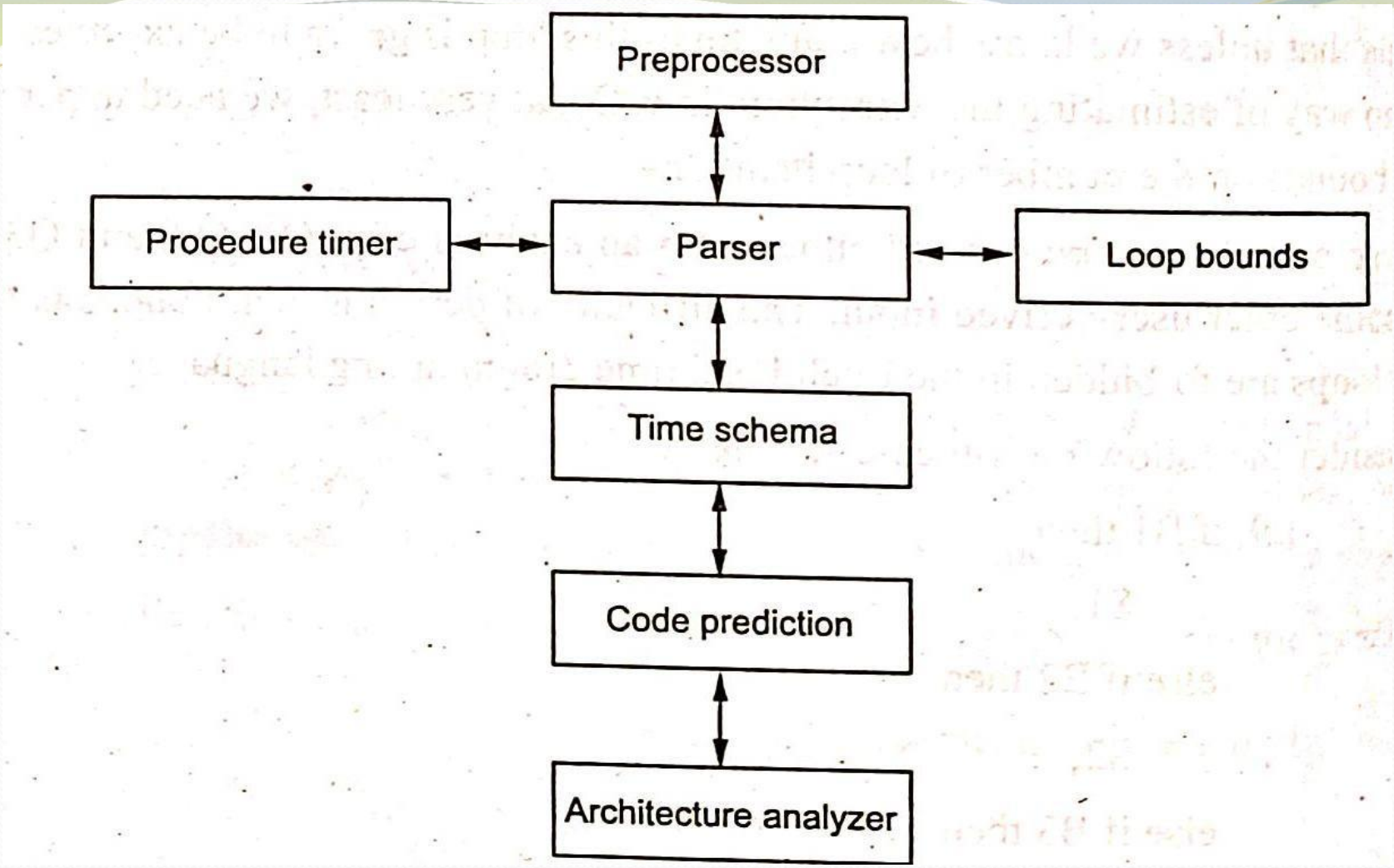
```
L9, if B1 then
        S1;
    else if B2 then
        S2;
    else if B3 then
        S3;
    else
        S4;
    end if;
```

In the case where B1 is true, the execution time is

$$T(B1) + T(S1) + T(JMP)$$

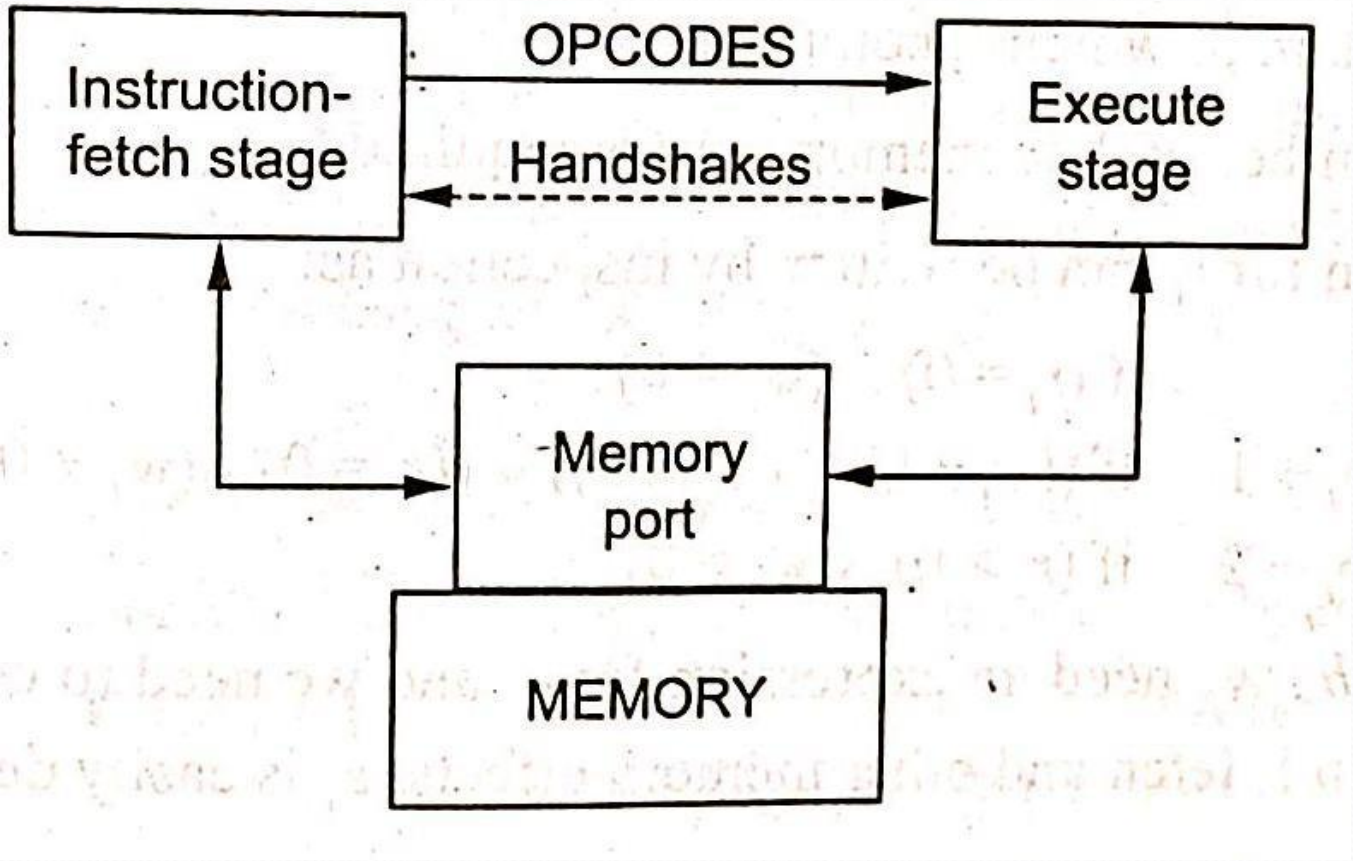
In the case where B1 is false but B2 is true, the execution time is

$$T(B1) + T(B2) + T(S2) + T(JMP)$$



Schematic of a timing estimation system

- Accounting of Pipeline



Two Stage pipeline

$$t_i = \begin{cases} b_i & \text{if } (r_i = 0) \wedge (w_i = 0) \\ b_i + 1 & \text{if } ((r_i \neq 0) \wedge (w_i = 0)) \vee ((r_i = 0) \wedge (w_i \neq 0)) \\ b_i + 2 & \text{if } (r_i > 0) \wedge (w_i > 0) \end{cases}$$

We can

$$b_i = \begin{cases} e_i + m\,(v_i - f_{i-1}) - h_{i-1} & \text{if case b1 applies} \\ e_i & \text{if Case b2.1 applies} \\ e_i + m - h_{i-1} & \text{if case b2.2 applies} \end{cases}$$

- Cache Memory
- Virtual Memory

4.3 Task Assignment and Scheduling

- A Task requires some execution time on a processor
- Also a task may required certain amount of memory or access to a bus
- Sometimes a resource must be exclusively held by a task
- In other cases resource may be exclusive or non exclusive depending on the operation to be performed on it

- Release Time
  - A task is a time at which all the data that are required to begin executing the Task are available
- Deadline
  - The deadline is the time by which the task must complete its execution
  - The deadline must be hard or soft
  - Task are classified as
    - Periodic
    - Sporadic
    - Aperiodic

- **Periodic**

  A task ti is periodic if it is released periodically. say every pi seconds pi is called the period of task Ti

- **Sporadic Task**
  - Sporadic task is a not periodic task, but may be invoked at irregular interval
  - Sporadic tasks are characterized by an upper bound on the rate at which they may be invoked

- **APeriodic Task**
- Tasks to be those tasks which are not periodic and which also have no upper bound on their invocation rate

- **Task Assignment / Schedule**
  - All task starts after the release time and complete before their deadline

  - A schedule may be
    - Precomputed(Offline scheduling)
    - Dynamically(Online Scheduling)

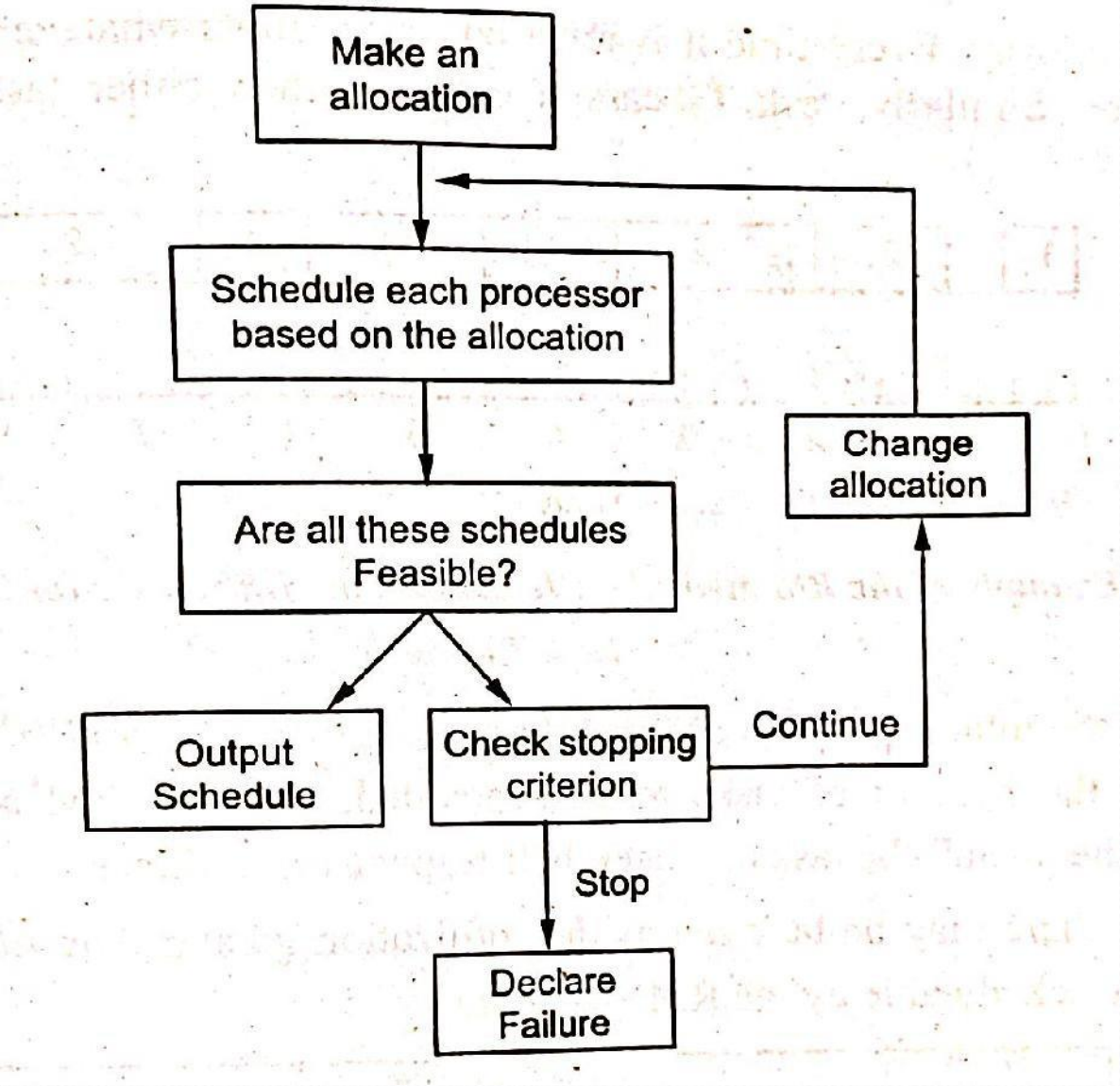
- Precomputed
  - Advance the operation with specification of periodic tasks will be run and slots for the sporadic / aperiodic tasks in the event that they are involved.

- Dynamically
  - Tasks are scheduled as they arrive in the system
  - The algorithm used in online scheduling must be fast and it takes to meet their deadlines is clearly useless
  - Two types priority algorithms are used
    - Static priority algorithm
    - Dynamic priority algorithm

- Static priority algorithm
  - Static priority algorithm assume that the task priority does not change within a mode
    - Example Rate monotonic algorithm

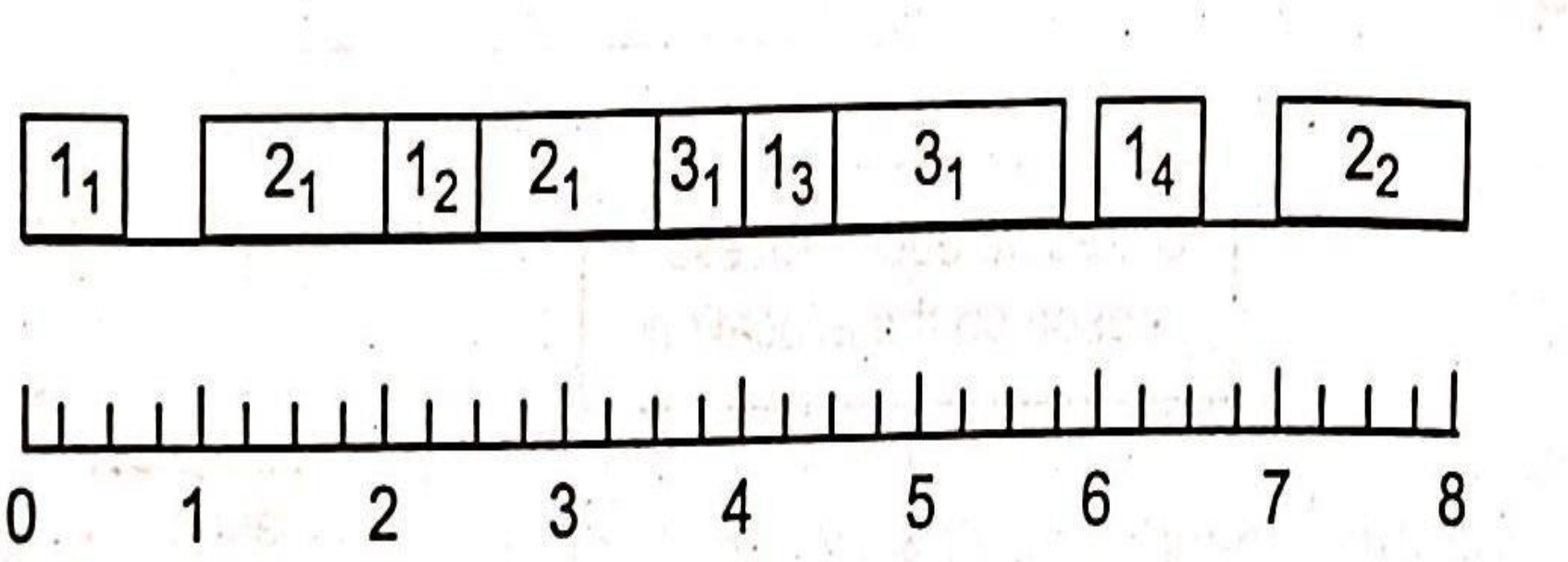
- Dynamic priority algorithm
  - algorithm assume that the task priority can change within a time
  - Example Earliest Deadline First (EDF)algorthim

- Classical uni-processor Scheduling Algorthim

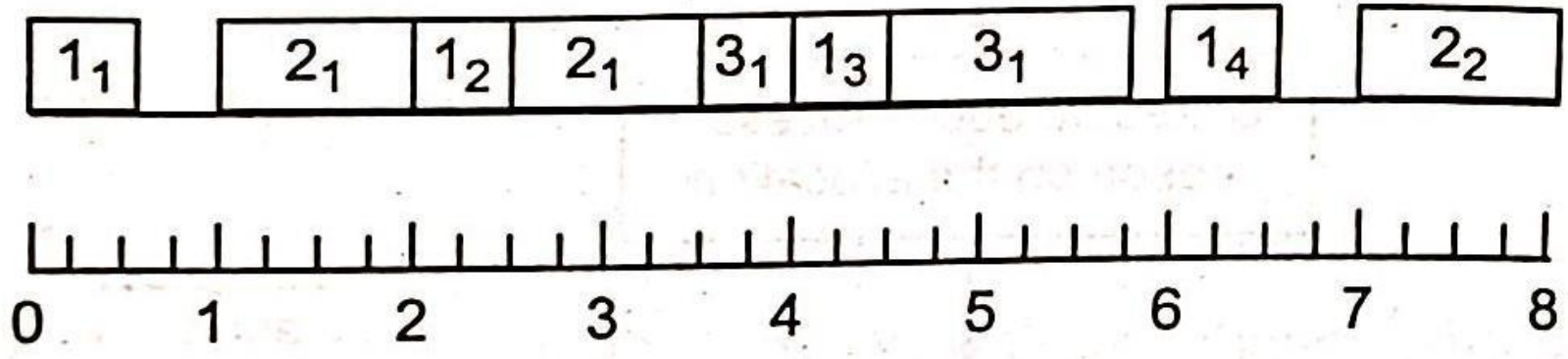


Example for Rate monotonic scheduling

There are three tasks, with $P_1 = 2$, $P_2 = 6$, $P_3 = 10$. The execution times are $e_1 = 0.5$, $e_2 = 2.0$, $e_3 = 1.75$ and $I_1 = 0$, $I_2 = 1$, $I_3 = 3$. Since $P_1 < P_2 < P_3$, task T_1 has highest priority. Every time it is released, it preempts whatever is running on the processor. Similarly, task T_3 cannot execute when either task T_1 or T_2 is unfinished.


1_1
2_1
1_2
2_1
3_1
1_3
3_1
1_4
2_2
0
1
2
3
4
5
6
7
8

There are three tasks, with $P_1 = 2$, $P_2 = 6$, $P_3 = 10$. The execution times are $e_1 = 0.5$, $e_2 = 2.0$, $e_3 = 1.75$ and $I_1 = 0$, $I_2 = 1$, $I_3 = 3$. Since $P_1 < P_2 < P_3$, task T_1 has highest priority. Every time it is released, it preempts whatever is running on the processor. Similarly, task T_3 cannot execute when either task T_1 or T_2 is unfinished.



UNIT V

PROCESSES AND OPERATING SYSTEMS

Introduction – Multiple tasks and multiple processes – Multirate systems- Preemptive real-time operating systems- Priority based scheduling- Interprocess communication mechanisms – Evaluating operating system performance- power optimization strategies for processes – Example Real time operating systems-POSIX-Windows-CE. Distributed embedded systems – MPSoCs and shared memory multiprocessors. – Design Example - Audio player, Engine control unit – Video accelerator.

INTRODUCTION

- Simple applications can be programmed on a microprocessor by writing a single piece of code.
- But for a complex application, multiple operations must be performed at widely varying times.
- Two fundamental abstractions that allow us to build complex applications on microprocessors.

1. ***Process***→ defines the state of an executing program
2. ***operating system (OS)***→provides the mechanism for switching execution between the processes.

MULTIPLE TASKS AND MULTIPLE PROCESSES

- Systems which are capable of performing multiprocessing known as multiple processor system.
- Multiprocessor system can execute multiple processes simultaneously with the help of multiple CPU.
- Multi-tasking→ The ability of an operating system to hold multiple processes in memory and switch the processor for executing one process.

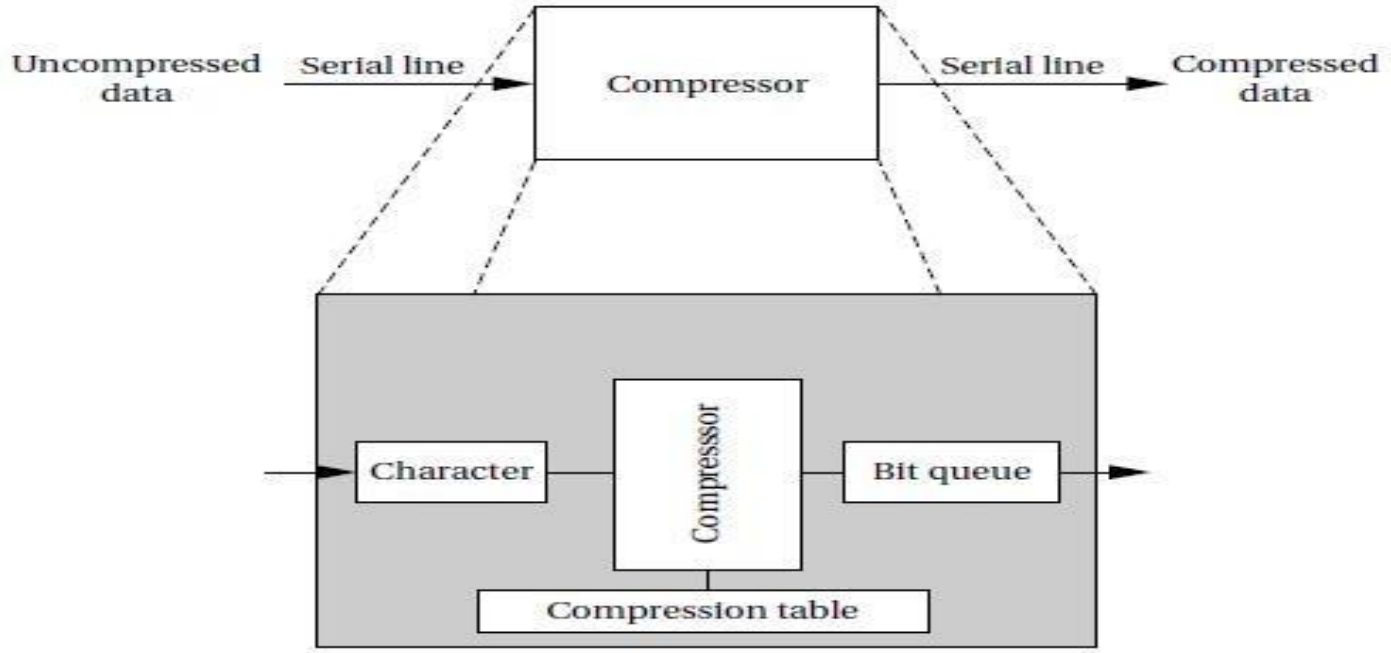
Tasks and Processes

- Task is nothing but different parts of functionality in a single system.
- Eg-Mobile Phones
- When designing a telephone answering machine, we can define recording a phone call ,answering a call and operating the user's control panel as distinct tasks, at different rates.
- Each application in a system is called a task.

Process

- A process is a single execution of a program.
- If we run the same program two different times, we have created two different processes.
- Each process has its own state that includes not only its registers but all of its memory.
- In some OSs, the memory management unit is used to keep each process in a separate address space.
- In others, particularly lightweight RTOSs, the processes run in the same address space.
- Processes that share the same address space are often called threads.

- This device is connected to serial ports on both ends.
- The input to the box is an uncompressed stream of bytes.
- The box emits a compressed string of bits, based on a compression table.



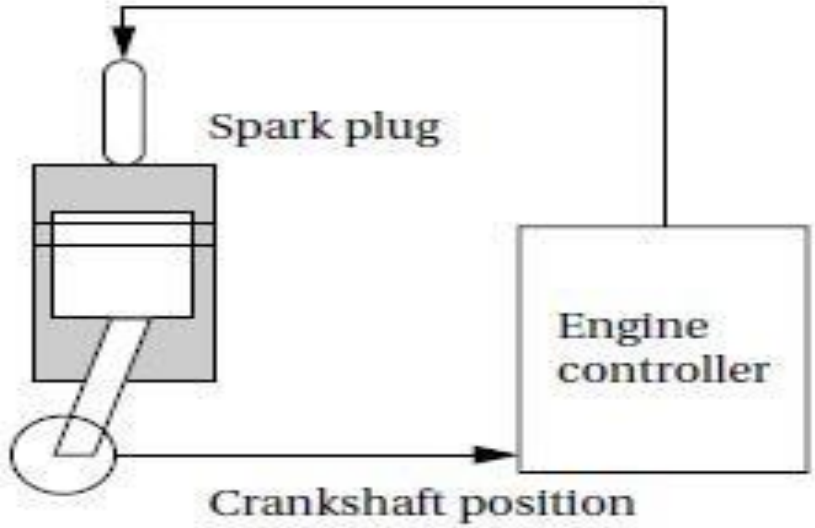
- Ex: compress data being sent to a modem.
- The program's need to receive and send data at different rates
- Eg→The program may emit 2 bits for the first byte and then 7 bits for the second byte— will obviously find itself reflected in the structure of the code.
- if we spend too much time in packaging and emitting output characters,we may drop an input character.

Asynchronous input

- Ex:A control panel on a machine provides a different type of rate.
- The control panel of the compression box include a compression mode button that disables or enables compression, so that the input text is passed through unchanged when compression is disabled.
- Sampling the button's state too slowly→ machine will miss a button depression entirely.
- Sampling it too frequently→ the machine will do incorrectly compress data.
- To solve this problem→ every n times the compression loop is executed.

Multi-rate Systems

- In operating system implementing code for satisfies timing requirements is more complex when multiple rates of computation must be handled.
- Multirate embedded computing systems→Ex: automobile engines, printers, and cell phones.
- In all these systems, certain operations must be executed periodically with its own rate.
- Eg→Automotive engine control



- The simplest automotive engine controllers, such as the ignition controller for a basic motorcycle engine, perform only one task—timing the firing of the spark plug, which takes the place of a mechanical distributor.

Spark Plug

- The spark plug must be fired at a certain point in the combustion cycle.

Microcontroller

- Using a microcontroller that senses the engine crankshaft position allows the spark timing to vary with engine speed.
- Firing the spark plug is a periodic process.

Engine controller

- Automobile engine controllers use additional sensors, including the gas pedal position and an oxygen sensor used to control emissions.
- They also use a multimode control scheme. one mode may be used for engine warm-up, another for cruise, and yet another for climbing steep hills.
- The engine controller takes a variety of inputs that determine the state of the engine.
- It then controls two basic engine parameters: the spark plug firings and the fuel/air mixture.

Task performed by engine controller unit

Variable	Time to move full range (ms)	Update period (ms)
Engine spark timing	300	2
Throttle	40	2
Airflow	30	4
Battery voltage	80	4
Fuel flow	250	10
Recycled exhaust gas	500	25
Set of status switches	100	50
Air temperature	seconds	500
Barometric pressure	seconds	1000
Spark/dwell	10	1
Fuel adjustments	80	4
Carburetor adjustments	500	25
Mode actuators	100	100

Timing Requirements on Processes

- Processes can have several different types of timing requirements based on the application.
- The timing requirements on a set of processes strongly depends on the type of scheduling.
- A scheduling policy must define the timing requirements that it uses to determine whether a schedule is valid.

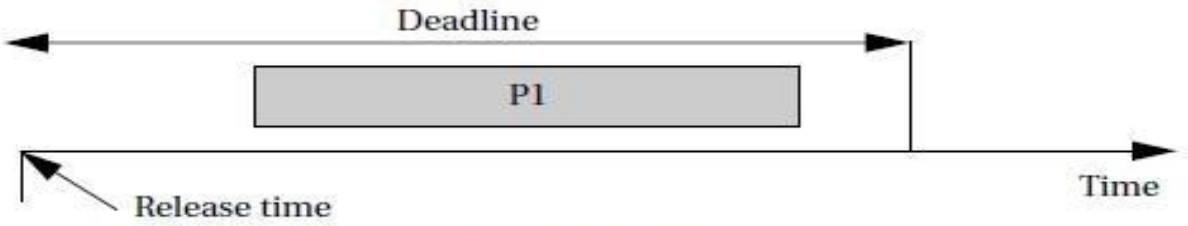
1. Release time→

- The time at which the process becomes ready to execute.
- simpler systems→ the process may become ready at the beginning of the period.
- sophisticated systems→ set the release time at the arrival time of certain data, at a time after the start of the period.

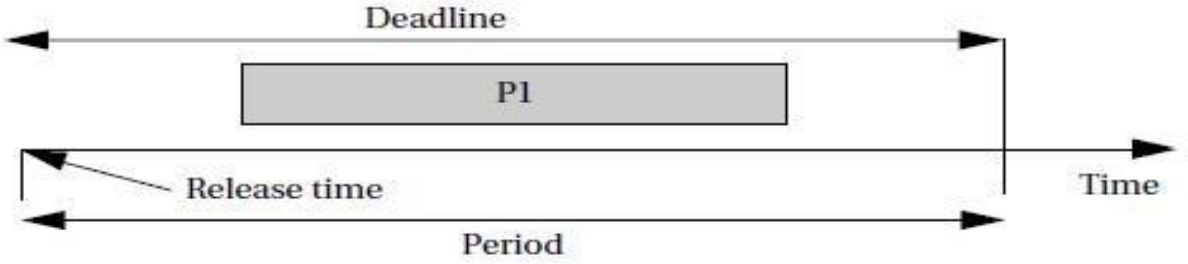
2. Deadline

- specifies when a computation must be finished.
- The deadline for an a periodic process is generally measured from the release time or initiation time.
- The deadline for a periodic process may occur at the end of the period.
- The period of a process is the time between successive executions.
- The process's rate is the inverse of its period.
- In a Multi rate system, each process executes at its own distinct rate.

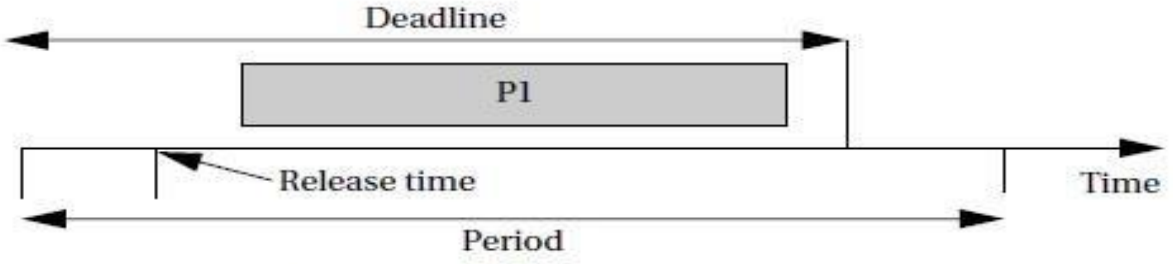
Example definitions of release times and deadlines



Aperiodic process

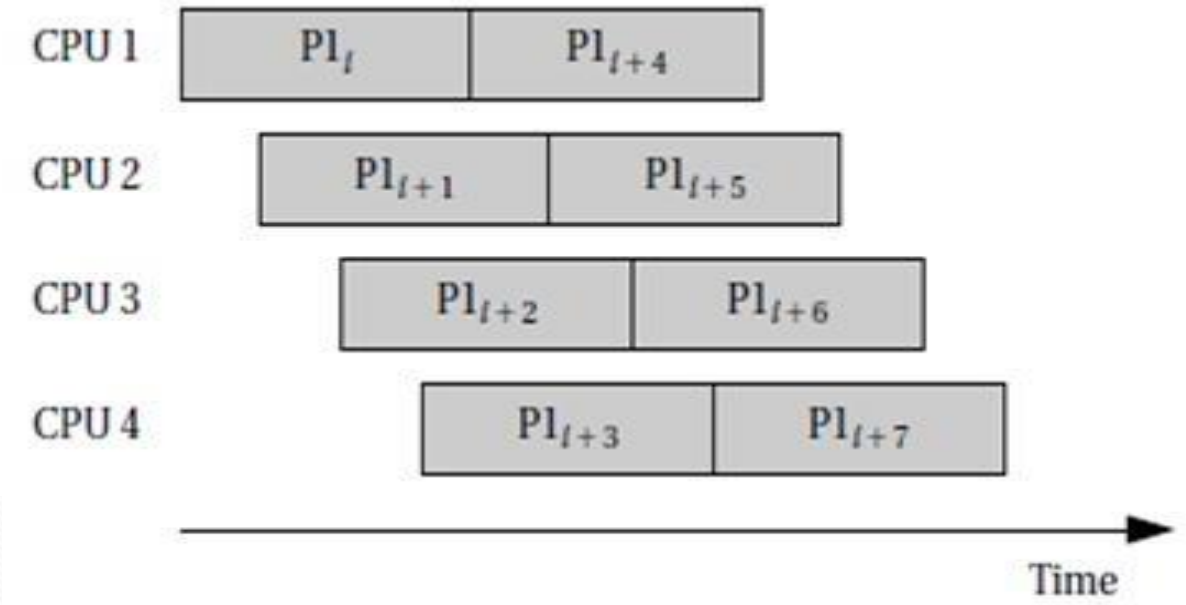


Periodic process initiated at start of period



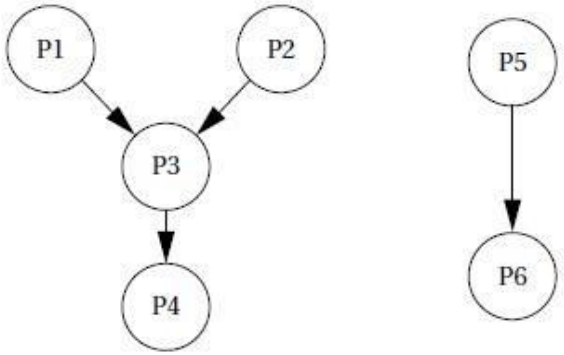
Periodic process released by event

A sequence of processes with a high initiation rate



- In this case, the initiation interval is equal to one fourth of the period.
- It is possible for a process to have an initiation rate less than the period even in single-CPU systems.
- If the process execution time is less than the period, it may be possible to initiate multiple copies of a program at slightly offset times.

Data dependencies among processes

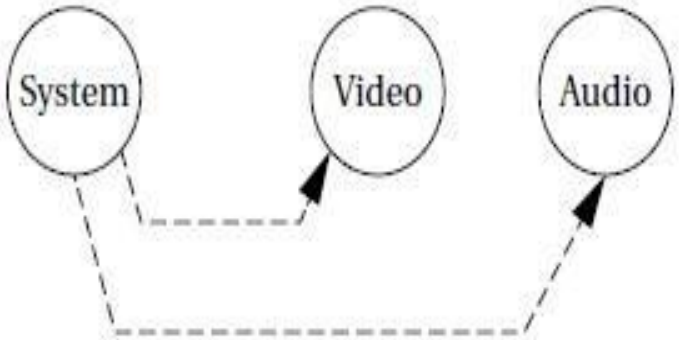


- The data dependencies define a partial ordering on process execution.
- P1 and P2 can execute in any order but must both complete before P3, and P3 must complete before P4.
- All processes must finish before the end of the period.

Directed Acyclic Graph (DAG)

- It is a directed graph that contains no cycles.
- The data dependencies must form a directed acyclic graph.
- A set of processes with data dependencies is known as a task graph.

Communication among processes at different rates (MPEG audio/Video)



- The system decoder process demultiplexes the audio and video data and distributes it to the appropriate processes.
- Missing Deadline
- Missing deadline in a multimedia system may cause an audio or video glitch.
- The system can be designed to take a variety of actions when a deadline is missed.

CPU Metrics

- CPU metrics are described by initiation time and completion time.
- Initiation time→It is the time at which a process actually starts executing on the CPU.
- Completion time→It is the time at which the process finishes its work.
- The CPU time of process i is called Ci .
- The CPU time is not equal to the completion time minus initiation time.
- The total CPU time consumed by a set of processes is

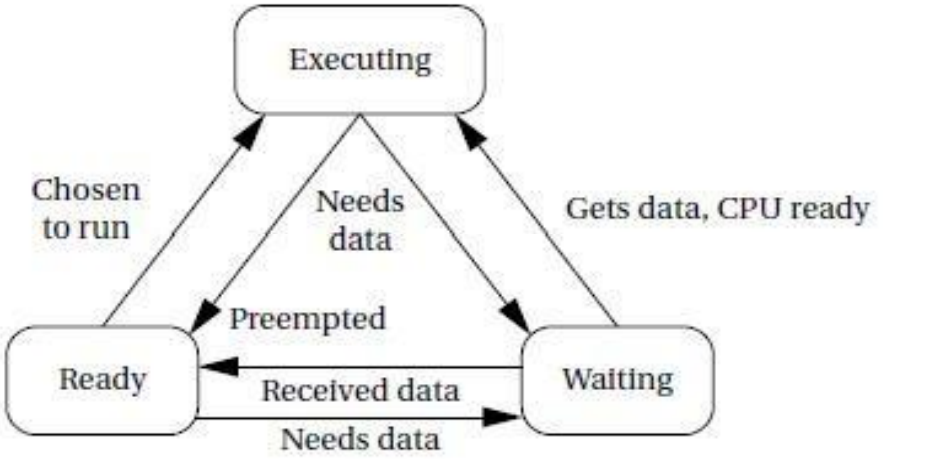
$$T = \sum_{1 \le i \le n} T_i.$$

- The simplest and most direct measure is utilization.

$$U = \frac{\text{CPU time for useful work}}{\text{total available CPU time}}.$$

Process State and Scheduling

- The first job of the OS is to determine that process runs next.
- The work of choosing the order of running processes is known as scheduling.
- There three basic scheduling ,such as waiting, ready and executing.



- A process goes into the waiting state when it needs data that it has finished all its work for the current period.
- A process goes into the ready state when it receives its required data, when it enters a new period.
- Finally a process can go into the executing state only when it has all its data, is ready to run, and the scheduler selects the process as the next process to run.

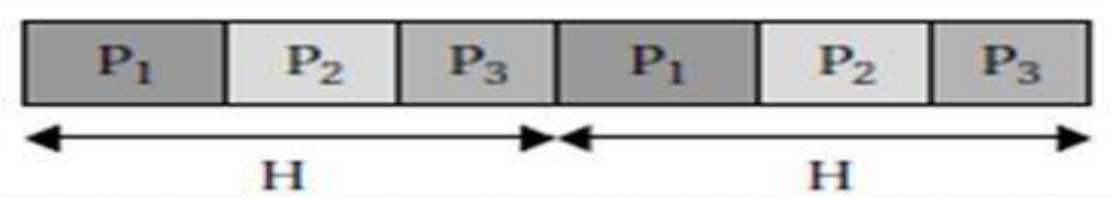
Scheduling Policies

- A scheduling policy defines how processes are selected for promotion from the ready state to the running state.
- Scheduling→Allocate time for execution of the processes in a system .
- For 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.
- Unrolled schedule →The complete schedule for the least-common multiple of the periods.

Types of scheduling

1. Cyclostatic scheduling or Time Division Multiple Access scheduling

- Schedule is divided into equal-sized time slots over an interval equal to the length of the hyperperiod *H*. (run in the same time slot)

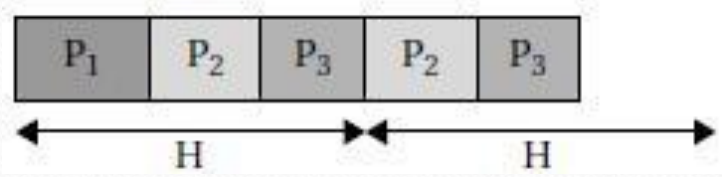


Two factors affect this scheduling

- The number of time slots used
- The fraction of each time slot that is used for useful work.

2)Round Robin-scheduling

- Uses the same hyper period as does cyclostatic.
- It also evaluates the processes in order.
- If a process does not have any useful work to do, the scheduler moves on to the next process in order to fill the time slot with useful work.



- All three processes execute during the first hyperperiod.
- During the second one, P1 has no useful work and is skipped so P3 is directly move on to the next process.

Scheduling overhead

- The execution time required to choose the next execution process, which is incurred in addition to any context switching overhead.

To calculate the utilization of CPU

Utilization of a set of processes

We are given three processes, their execution times, and their periods:

Process	Period	Execution time
P1	1.0×10^{-3}	1.0×10^{-4}
P2	1.0×10^{-3}	2.0×10^{-4}
P3	5.0×10^{-3}	3.0×10^{-4}

The least common multiple of these periods is 5×10^{-3} s.

We can now determine the utilization over the hyperperiod:

$$U = \frac{5.1 \times 10^{-4} + 5.2 \times 10^{-4} + 1.3 \times 10^{-4}}{5 \times 10^{-3}} = 0.36$$

This is well below our maximum utilization of 1.0.

Preemptive Real-Time Operating Systems(RTOS)

- A pre emptive OS →solves the fundamental problem in multitasking system.
- It executes processes based upon timing requirements provided by the system designer.
- To meet timing constraints accurately is to build a preemptive OS and to use priorities to control what process runs at any given time.

Preemption

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

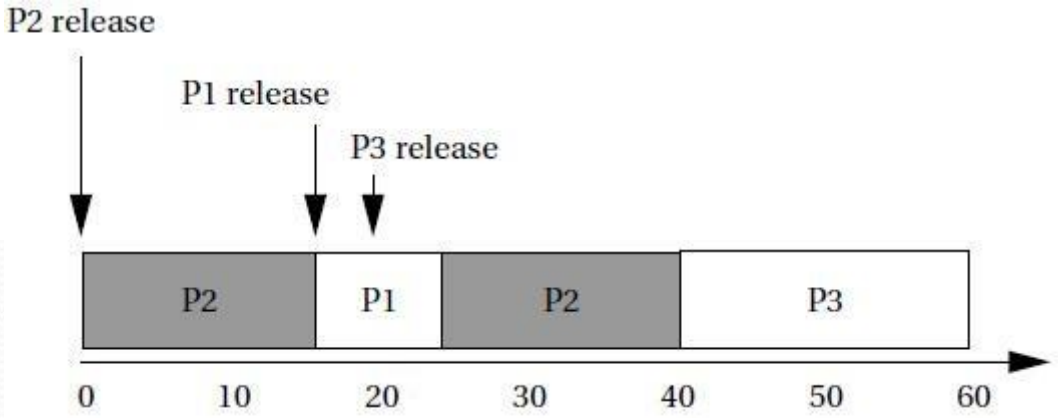
Kernel

- It is the part of the OS that determines what process is running.
- The kernel is activated periodically by the timer.
- It determines what process will run next and causes that process to run.

Priorities

- Based on the priorities →kernel can do the processes sequentially.
- which ones actually want to execute 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.

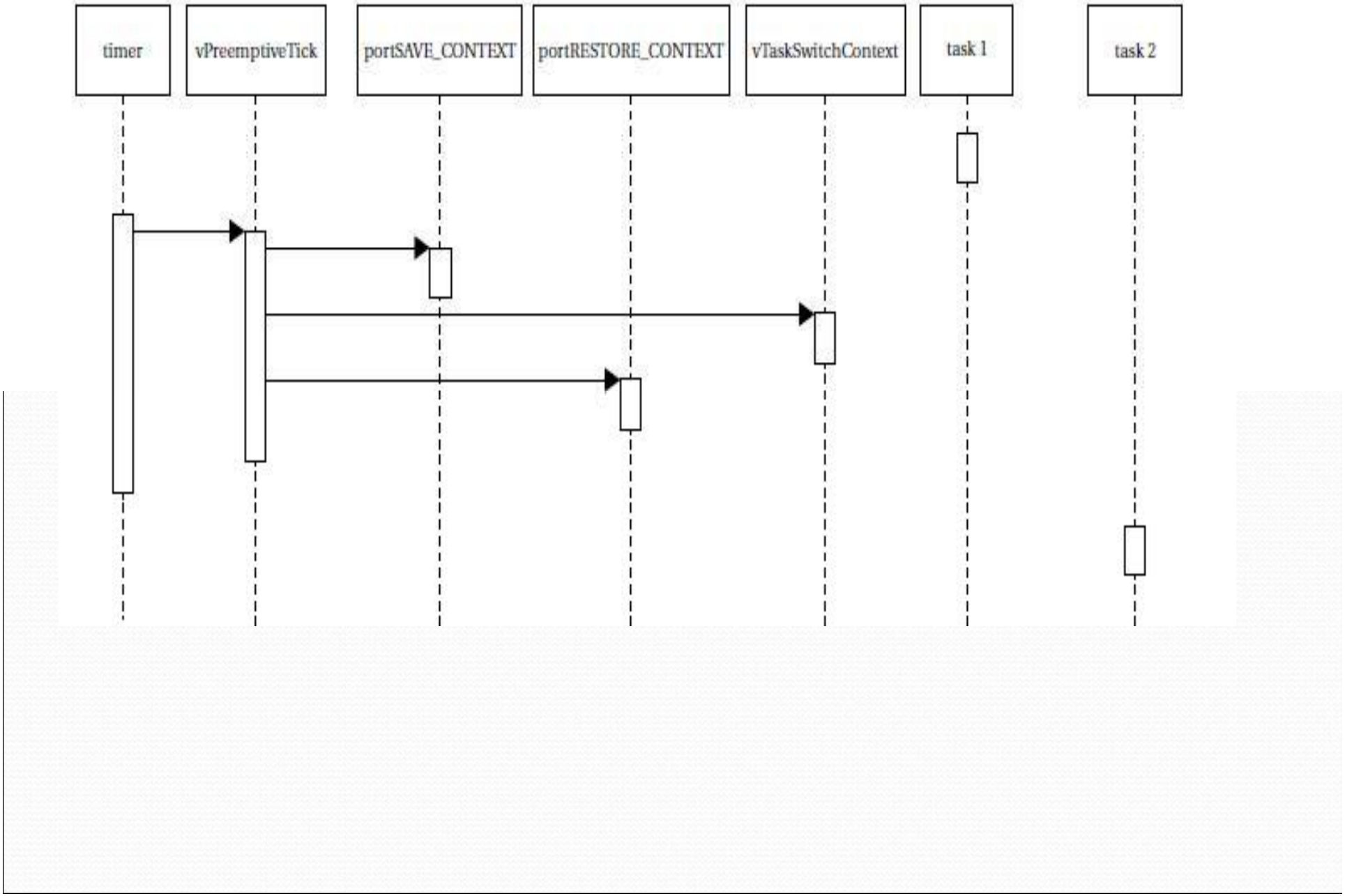
Process	Priority	Execution time
P1	1	10
P2	2	30
P3	3	20



•When the system begins execution,P2 is the only ready process, so it is selected for execution.
•At T=15, P1 becomes ready; it preempts P2 because p1 has a higher priority, so it execute immediately
•P3's data arrive at time 18, it has lowest priority.
•P2 is still ready and has higher priority than P3.
•Only after both P1 and P2 finish can P3 execute

- 5.4.4) Context Switching

- To understand the basics of a context switch, let's assume that the set of 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 application tasks, the hardware timer, and all the functions in the kernel that are involved in the context switch.
- vPreemptiveTick() → it is called when the timer ticks.
- portSAVE_CONTEXT()→ swaps out the current task context.
- vTaskSwitchContext ( ) →chooses a new task.
- portRESTORE_CONTEXT()→ swaps in the new context


timer
vPreemptiveTick
portSAVE_CONTEXT
portRESTORE_CONTEXT
vTaskSwitchContext
task 1
task 2

PRIORITY-BASED SCHEDULING

- Operating system is to allocate resources in the computing system based on the priority.
- After assigning priorities, the OS takes care of the rest by choosing the highest-priority ready process.
- There are two major ways to assign priorities.
- **Static priorities→** that do not change during execution
- **Dynamic priorities→** that do change during execution
- Types of scheduling process

1. Rate-Monotonic Scheduling
2. Earliest-Deadline-First Scheduling

Rate-Monotonic Scheduling(RMS)

- Rate-monotonic scheduling (RMS)→ is one of the first scheduling policies developed for real-time systems.
- RMS is a static scheduling policy.
- It assigns fixed priorities are sufficient to efficiently schedule the processes in many situations.

RMS is known as rate-monotonic analysis (RMA), as summarized below.

- All processes run periodically on a single CPU.
- Context switching time is ignored.
- There are no data dependencies between processes.
- The execution time for a process is constant.
- All deadlines are at the ends of their periods.
- The highest-priority ready process is always selected for execution.
- Priorities are assigned by rank order of period, with the process with the shortest period being assigned the highest priority.

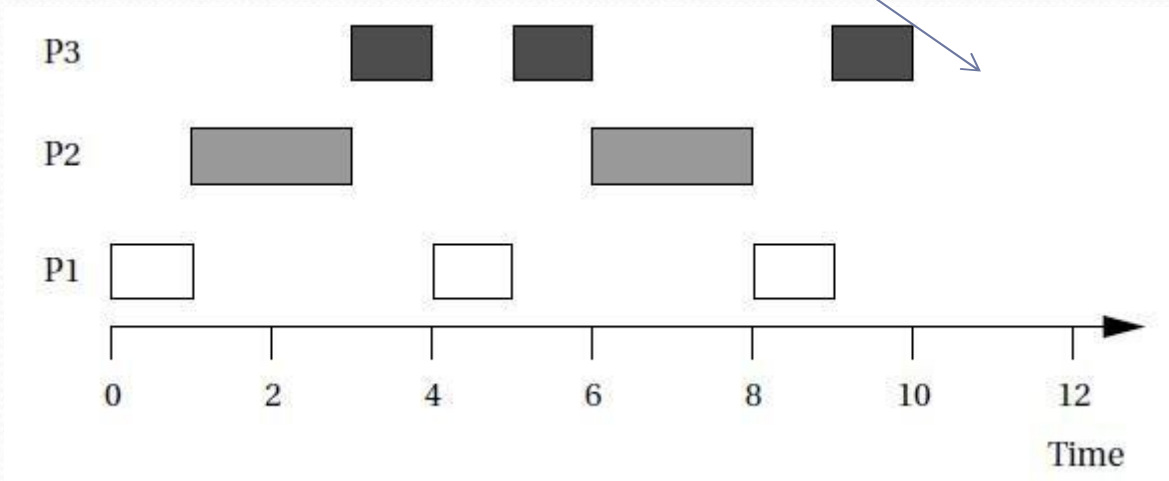
Example-Rate-monotonic scheduling

- set of processes and their characteristics

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12

- According to RMA → Assign highest priority for least execution period.
- Hence P1 the highest priority, P2 the middle priority,and P3 the lowest priority.
- First execute P1 then P2 and finally P3.(T1>T2>T3)
- After assigning priorities, construct a time line equal in length to hyper period, which is 12 in this case.

- Every 4 time intervals P1 executes 1 units.(Execution time intervals for P1 0-4,4-8,8-12)
- Every 6 time intervals P2 executes 2 units. .(Execution time intervals for P2 0-6,6-12)
- Every 12 intervals P3 executes 3 units. .(Execution time intervals for P3 0-12)
- Time interval from 10-12 no scheduling available because no process will be available for execution. All process are executed already.



- P1 is the highest-priority process, it can start to execute immediately.
- After one time unit, P1 finishes and goes out of the ready state until the start of its next period.
- At time 1, P2 starts executing as the highest-priority ready process.
- At time 3, P2 finishes and P3 starts executing.
- P1's next iteration starts at time 4, at which point it interrupts P3.
- P3 gets one more time unit of execution between the second iterations of P1 and P2, but P3 does not get to finish until after the third iteration of P1.
- Consider the following different set of execution times.

Process	Execution time	Period
P1	2	4
P2	3	6
P3	3	12

- In this case, Even though each process alone has an execution time significantly less than its period, combinations of processes can require more than 100% of the available CPU cycles.
- During one 12 time-unit interval, we must execute P1 -3 times, requiring 6 units of CPU time; P2 twice, costing 6 units and P3 one time, costing 3 units.
- The total of 6 + 6 + 3 = 15 units of CPU time is more than the 12 time units available, clearly exceeding the available CPU capacity(12units).

RMA priority assignment analysis

- Response time→ The time at which the process finishes.
- Critical instant→The instant during execution at which the task has the largest response time.
- Let the periods and computation times of two processes P1 and P2 be τ1, τ2 and *T1, T2, with* τ *1* < τ *2.*
- let P1 have the higher priority. In the worst case we then execute P2 once during its period and as many iterations of P1 as fit in the same interval.
- Since there are τ2/ τ1 *iterations of P1 during a single* period of P2.
- The required constraint on CPU time, ignoring context switching overhead, is

$$\left\lceil \frac{\tau_2}{\tau_1} \right\rceil T_1 + T_2 \leq \tau_2.$$

- we give higher priority to P2, then execute all of P2 and all of P1 in one of P1's periods in the worst case.

$$T_1 + T_2 \leq \tau_1.$$

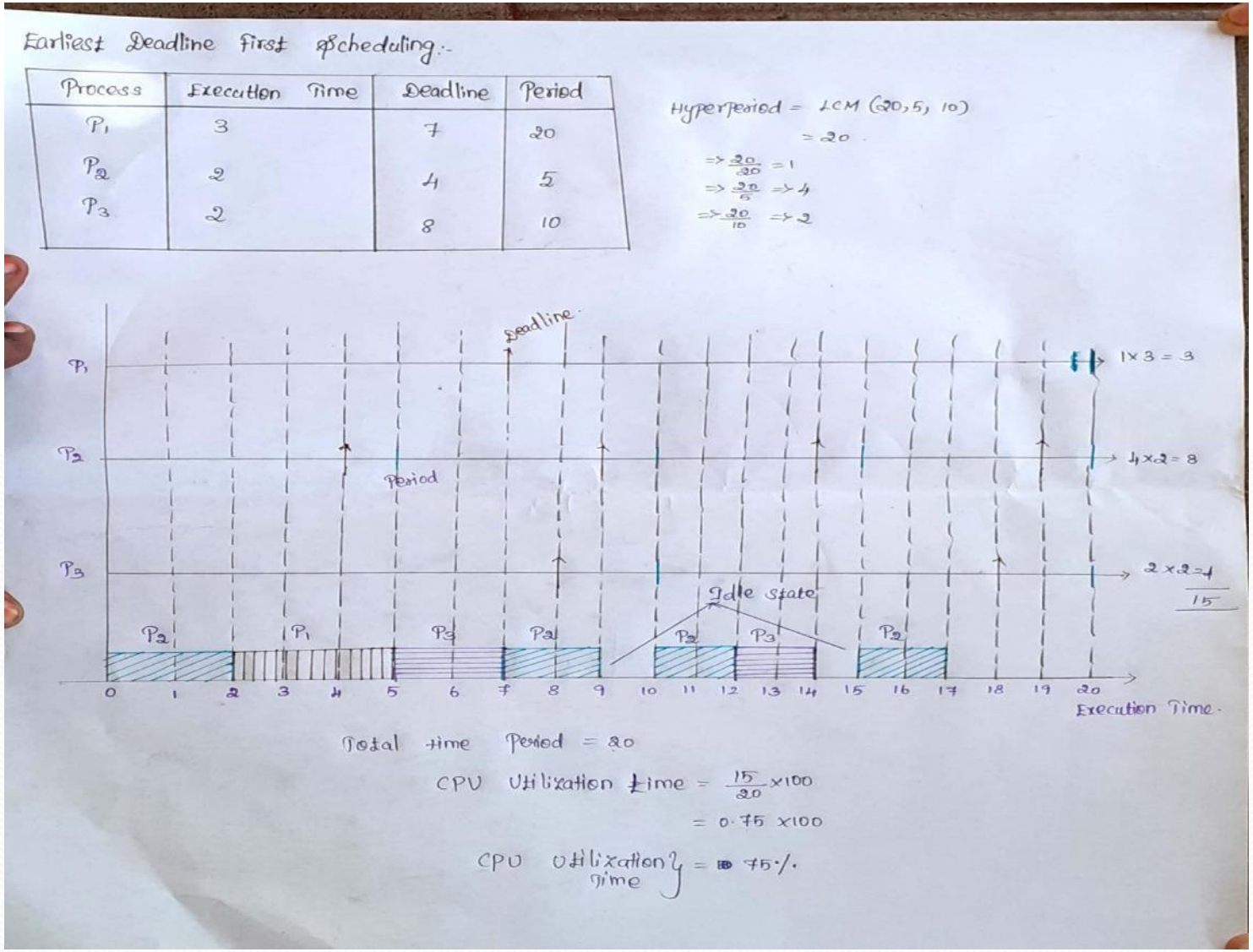
- Total CPU utilization for a set of *n tasks is* $U = \sum_{i=1}^{n} \frac{T_i}{\tau_i}.$

Earliest-Deadline-First Scheduling(EDF)

- Earliest deadline first (EDF)→ is a dynamic priority scheme.
- It changes process priorities during execution based on initiation times.
- As a result, it can achieve higher CPU utilizations than RMS.
- The EDF policy is also very simple.
- It assigns priorities in order of deadline.
- Assign highest priority to a process who has Earliest deadline.
- Assign lowest priority to a process who has farthest deadline.
- After assigning scheduling procedure, the highest-priority process is chosen for execution.
- Consider the following Example

- Hyper-period is 60

Dead line Table



- There is one time slot left at *t= 30, giving a CPU utilization of 59/60.*
- EDF can achieve 100% utilization
- **RMS vs. EDF**

	RMS	EDF
(*i*)	Less utilization of CPU	More utilization of CPU
(*ii*)	It is static priority	It is dynamic priority
(*iii*)	It can diagnose the overload	It cannot diagnose the overload
(*iv*)	It very easier to ensure all the deadline	Much difficult compare to RMS
(*v*)	While CPU utilization is less, this gives less problem to complete the deadline.	It is more Problematic

Ex:Priority inversion

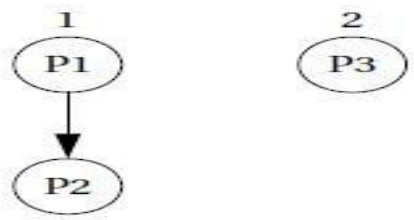
- Low-priority process blocks execution of a higher priority process by keeping hold of its resource.

Consider a system with two processes

- Higher-priority P1 and the lower-priority P2.
- Each uses the microprocessor bus to communicate to peripherals.
- When P2 executes, it requests the bus from the operating system and receives it.
- If P1 becomes ready while P2 is using the bus, the OS will preempt P2 for P1, leaving P2 with control of the bus.
- When P1 requests the bus, it will be denied the bus, since P2 already owns it.
- Unless P1 has a way to take the bus from P2, the two processes may deadlock.

Eg:Data dependencies and scheduling

- Data dependencies imply that certain combinations of processes can never occur. Consider the simple example.



Task graph

Task	Deadline
1	10
2	8

Task rates

Process	CPU time
P1	2
P2	1
P3	4

Execution times

- We know that P1 and P2 cannot execute at the same time, since P1 must finish before P2 can begin.
- P3 has a higher priority, it will not preempt both P1 and P2 in a single iteration.
- If P3 preempts P1, then P3 will complete before P2 begins.
- if P3 preempts P2, then it will not interfere with P1 in that iteration.
- Because we know that some combinations of processes cannot be ready at the same time, worst-case CPU requirements are less than would be required if all processes could be ready simultaneously.

Inter-process communication mechanisms

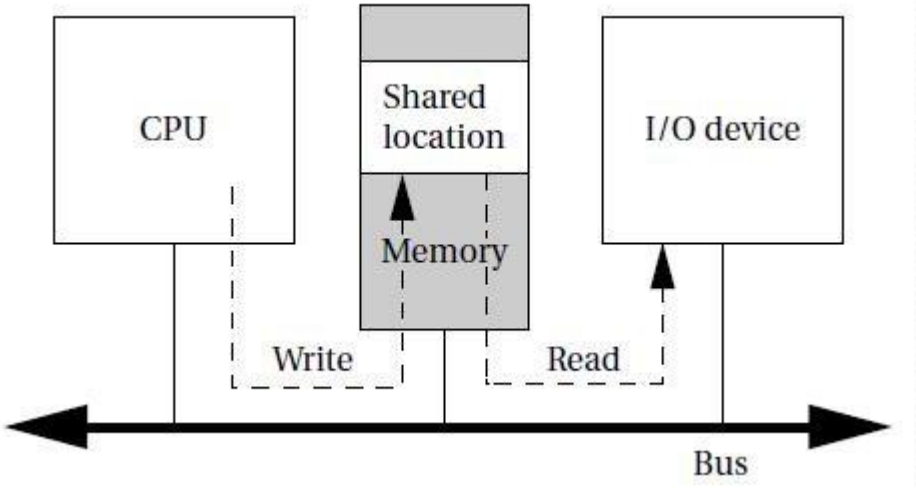
- It is provided by the operating system as part of the process abstraction.
- **Blocking Communication→** The process goes into the waiting state until it receives a response
- **Non-blocking Communication→**It allows a process to continue execution after sending the communication.

Types of inter-process communication

1. Shared Memory Communication
2. Message Passing
3. Signals

Shared Memory Communication

- The communication between inter-process is used by bus-based system.
- CPU and an I/O device, communicate through a shared memory location.
- The software on the CPU has been designed to know the address of the shared location.
- The shared location has also been loaded into the proper register of the I/O device.
- If CPU wants to send data to the device, it writes to the shared location.
- The I/O device then reads the data from that location.
- The read and write operations are standard and can be encapsulated in a procedural interface.



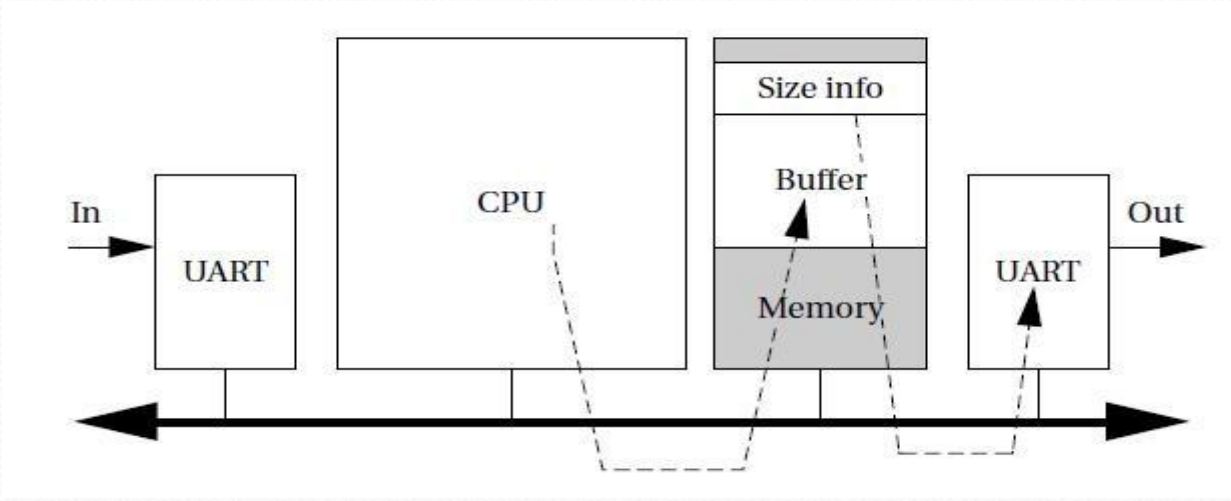
- CPU and the I/O device want to communicate through a shared memory block.
- There must be a flag that tells the CPU when the data from the I/O device is ready.
- The flag value of 0 when the data are not ready and 1 when the data are ready.
- If the flag is used only by the CPU, then the flag can be implemented using a standard memory write operation.
- If the same flag is used for bidirectional signaling between the CPU and the I/O device, care must be taken.

Consider the following scenario to call flag

1. CPU reads the flag location and sees that it is 0.
2. I/O device reads the flag location and sees that it is 0.
3. CPU sets the flag location to 1 and writes data to the shared location.
4. I/O device erroneously sets the flag to 1 and overwrites the data left by the CPU.

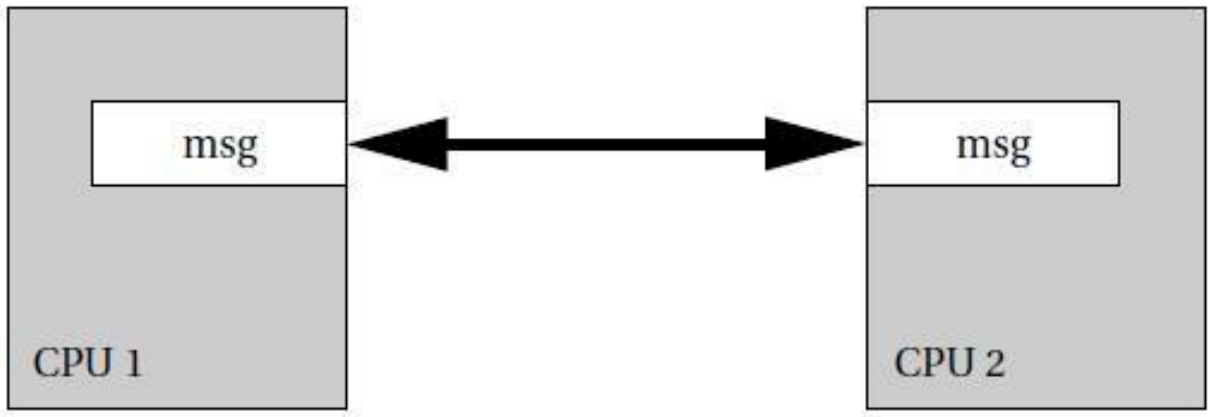
Ex: Elastic buffers as shared memory

- The text compressor is a good example of a shared memory.
- The text compressor uses the CPU to compress incoming text, which is then sent on a serial line by a UART.
- The input data arrive at a constant rate and are easy to manage.
- But 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 into 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.
- 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 overrun the buffer.



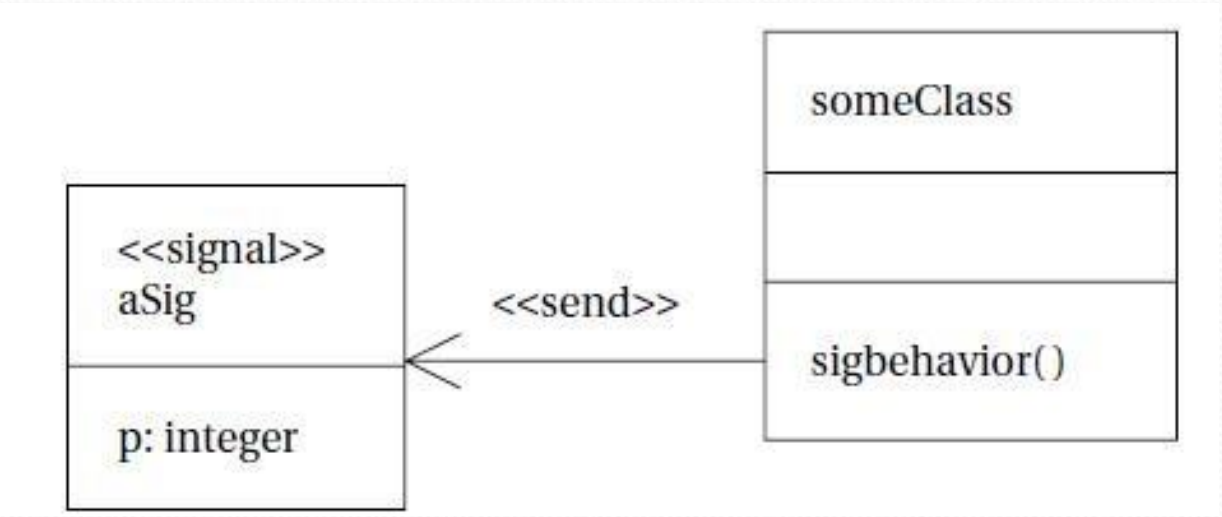
Message Passing

- Here each communicating entity has its own message send/receive unit.
- The message is not stored on the communications link, but rather at the senders/ receivers at the end points.
- Ex:Home control system
- It has one microcontroller per household device—lamp, thermostat, faucet, appliance.
- The devices must communicate relatively infrequently.
- Their physical separation is large enough that we would not naturally think of them as sharing a central pool of memory.
- Passing communication packets among the devices is a natural way to describe coordination between these devices.



Signals

- Generally signal communication used in Unix .
- A signal is analogous to an interrupt, but it is entirely a software creation.
- A signal is generated by a process and transmitted to another process by the OS.
- A UML signal is actually a generalization of the Unix signal.
- Unix signal carries no parameters other than a condition code.
- UML signal is an object, carry parameters as object attributes.
- The sigbehavior( ) →behavior of the class is responsible for throwing the signal, as indicated by<<send>>.
- The signal object is indicated by the <<signal>>



Evaluating operating system performance

- Analysis of scheduling policies is made by the following 4 assumptions
- Assumed that context switches require zero time. Although it is often reasonable to neglect context switch time when it is much smaller than the process execution time, context switching can add significant delay in some cases.
- We have largely ignored interrupts. The latency from when an interrupt is requested to when the device's service is complete is a critical parameter of real time performance.
- We have assumed that we know the execution time of the processes.
- We probably determined worst-case or best-case times for the processes in isolation.

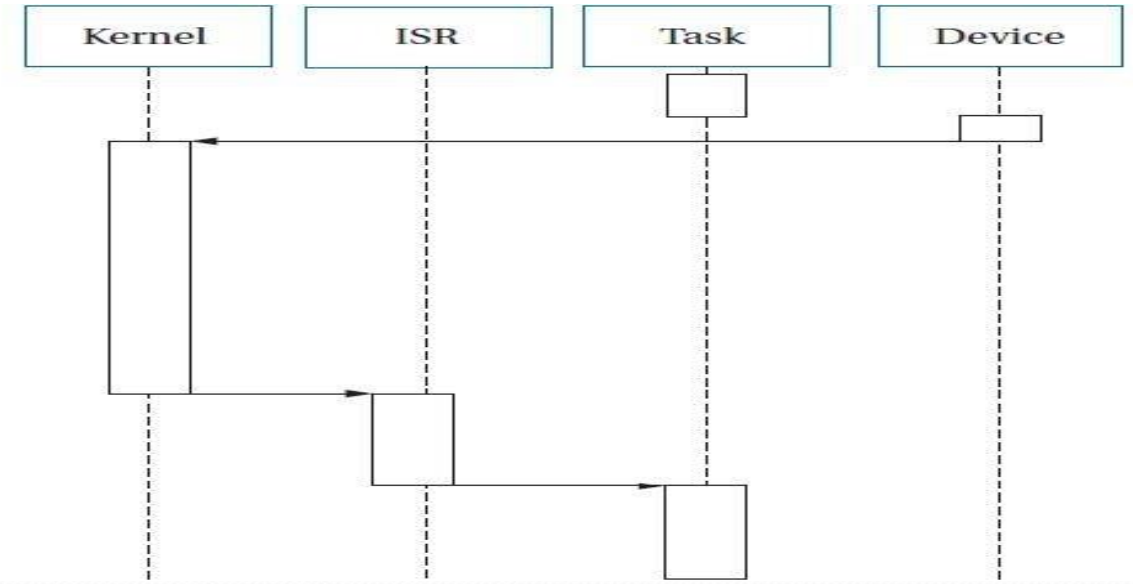
Context switching time

It depends on following factors

- The amount of CPU context that must be saved.
- Scheduler execution time.

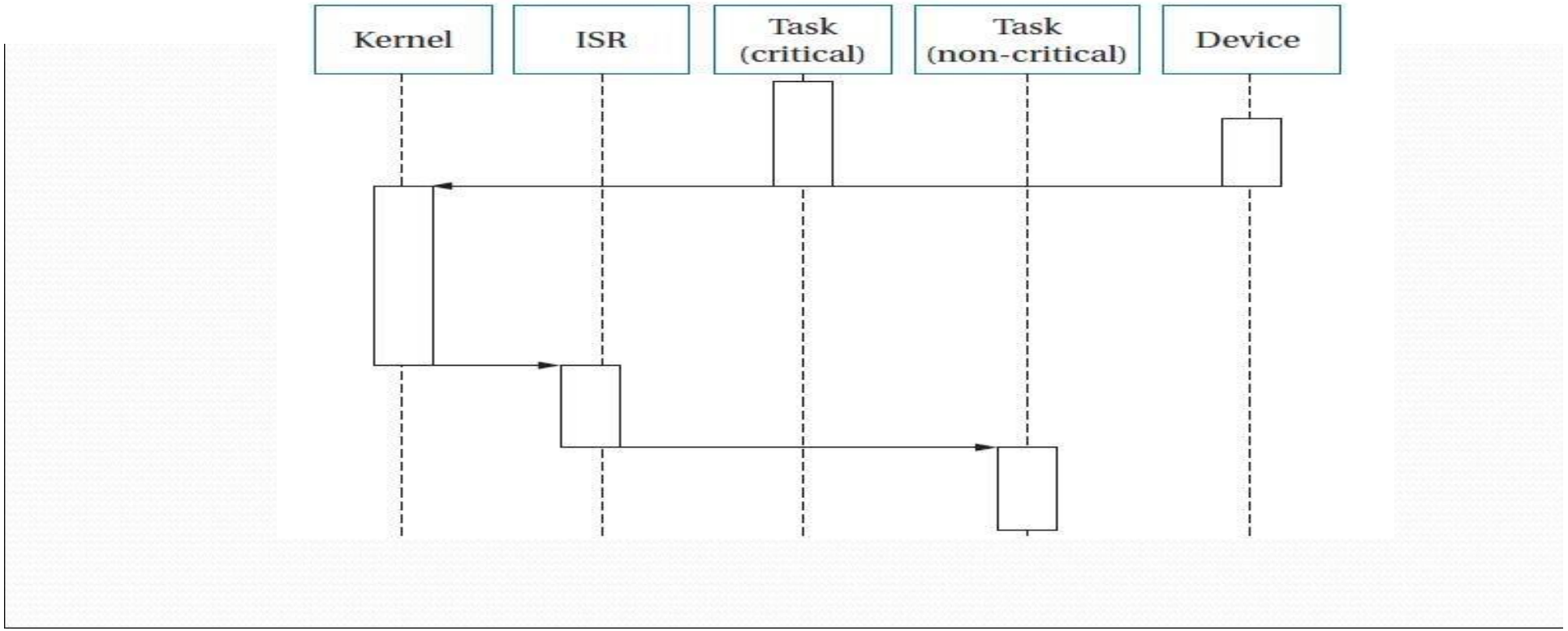
Interrupt latency

- Interrupt latency →It is the duration of time from the assertion of a device interrupt to the completion of the device's requested operation.
- Interrupt latency is critical because data may be lost when an interrupt is not serviced in a timely fashion.



- A task is interrupted by a device.
- The interrupt goes to the kernel, which may need to finish a protected operation.
- Once the kernel can process the interrupt, it calls the interrupt service routine (ISR), which performs the required operations on the device.
- Once the ISR is done, the task can resume execution.

- Several factors in both hardware and software affect interrupt latency:
- The processor interrupt latency
- The execution time of the interrupt handler
- Delays due to RTOS scheduling
- RTOS delay the execution of an interrupt handler in two ways.
- Critical sections and interrupt latency
- Critical sections in the kernel will prevent the RTOS from taking interrupts.
- Some operating systems have very long critical sections that disable interrupt handling for very long periods.



- If a device interrupts during a critical section, that critical section must finish before the kernel can handle the interrupt.
- The longer the critical section, the greater the potential delay.
- Critical sections are one important source of scheduling jitter because a device may interrupt at different points in the execution of processes and hit critical sections at different points.

Interrupt priorities and interrupt latency

- A higher-priority interrupt may delay a lower-priority interrupt.
- A hardware interrupt handler runs as part of the kernel, not as a user thread.
- The priorities for interrupts are determined by hardware.
- Any interrupt handler preempts all user threads because interrupts are part of the CPU's fundamental operation.
- We can reduce the effects of hardware preemption by dividing interrupt handling into two different pieces of code.
- **Interrupt service handler (ISH)→** performs the minimal operations required to respond to the device.
- **Interrupt service routine (ISR)→** Performs updating user buffers or other more complex operation.

- RTOS performance evaluation tools
- Some RTOSs provide simulators or other tools that allow us to view the operation of the processes,context switching time, interrupt response time, and other overheads.

Windows CE provides several performance analysis toolsAn instrumentation routine in the kernel that measures both interrupt serviceroutine and interrupt service thread latency.

- OS Bench measures the timing of operating system tasks such as critical section access, signals, and so on
- Kernel Tracker provides a graphical user interface for RTOS events.

Power optimization strategies for processes

- A power management policy is a strategy for determining when to perform certain power management operations.
- The system can be designed based on the static and dynamic power management mechanisms.

Power saving straegies

- Avoiding a power-down mode can cost unnecessary power.
- Powering down too soon can cause severe performance penalties.
- Re-entering run mode typically costs a considerable amount of time.
- A straightforward method is to power up the system when a request is received.

Predictive shutdown

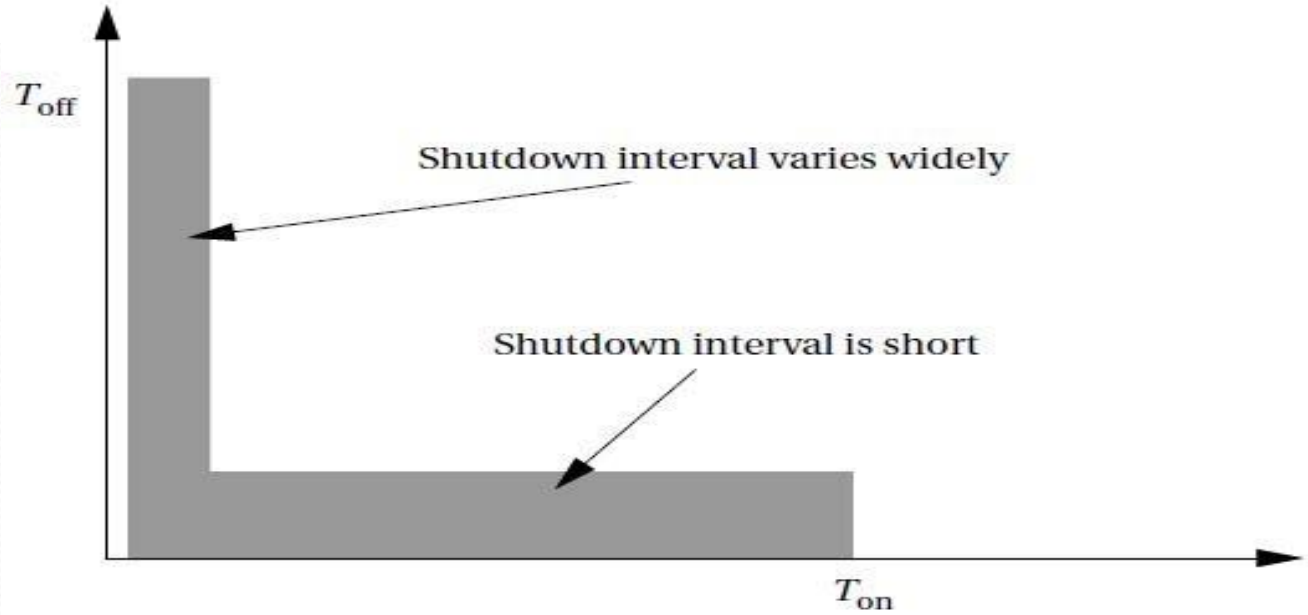
- The goal is to predict when the next request will be made and to start the system just before that time, saving the requestor the start-up time.
- Make guesses about activity patterns based on a probabilistic model of expected behavior.

This can cause two types of problems

- The requestor may have to wait for an activity period.
- In the worst case,the requestor may not make a deadline due to the delay incurred by system

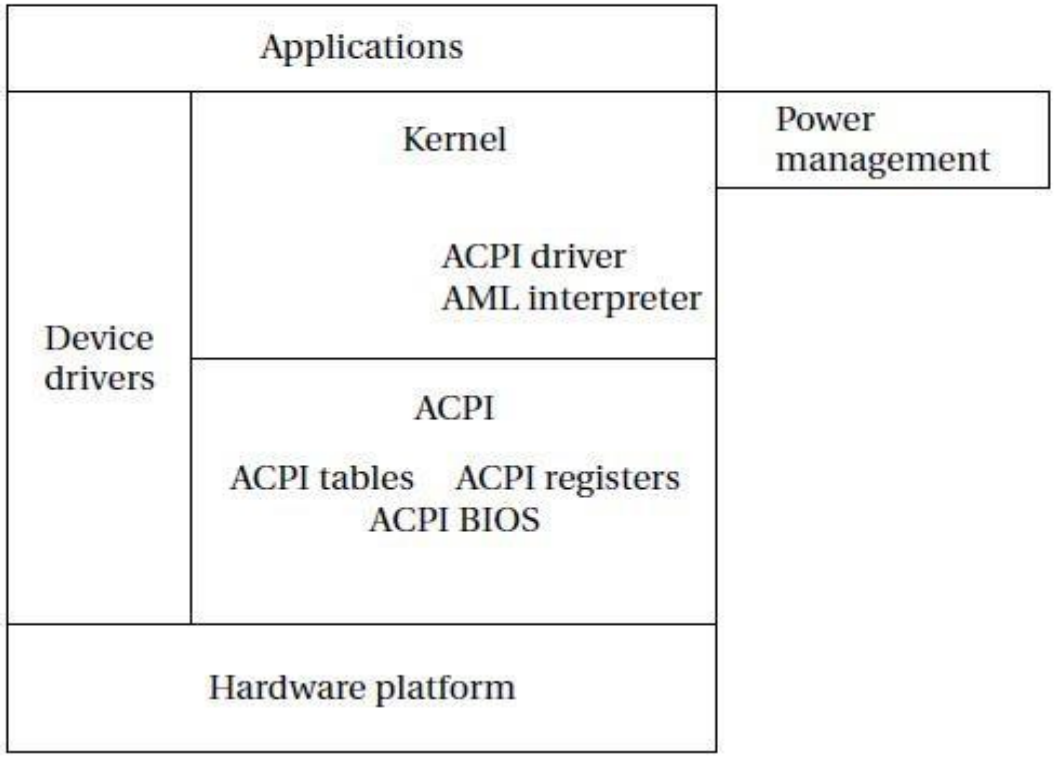
An L-shaped usage distribution

- A very simple technique is to use fixed times.
- If the system does not receive inputs during an interval of length *Ton, it shuts down.*
- Powered-down system waits for a period *Toff before* returning to the power-on mode.
- In this distribution, the idle period after a long active period is usually very short, and the length of the idle period after a short active period is uniformly distributed.
- Based on this distribution, shutdown when the active period length was below a threshold, putting the system in the vertical portion of the *L distribution.*



Advanced Configuration and Power Interface (ACPI)

- It is an open industry standard for power management services.
- It is designed to be compatible with a wide variety of OSs.
- A decision module →determines power management actions.



ACPI supports the following five basic global power states.

1. G3, the mechanical off state, in which the system consumes no power.
2. G2, the soft off state, which requires a full OS reboot to restore the machine to working condition. This state has four sub-states:

- S1, a low wake-up latency state with no loss of system context
- S2, a low wake-up latency state with a loss of CPU and system cache state
- S3, a low wake-up latency state in which all system state except for main
- memory is lost.

S4, the lowest-power sleeping state, in which all devices are turned off.

3. G1, the sleeping state, in which the system appears to be off.
4. G0, the working state, in which the system is fully usable.
5. The legacy state, in which the system does not comply with ACPI.

Example Real time operating systems

POSIX

- POSIX is a Unix operating system created by a standards organization.
- POSIX-compliant operating systems are source-code compatible.
- Application can be compiled and run without modification on a new POSIX platform.
- It has been extended to support real time requirements.
- Many RTOSs are POSIX-compliant and it serves as a good model for basic RTOS techniques.
- The Linux operating system has a platform for embedded computing.
- Linux is a POSIX-compliant operating system that is available as open source.
- Linux was not originally designed for real-time operation .
- Some versions of Linux may exhibit long interrupt latencies,
- To improve interrupt latency,A dual-kernel approach uses a specialized kernel, the co-kernel, for real-time processes **and** the standard kernel for non-real-time processes.

Process in POSIX

- A new process is created by making a copy of an existing process.
- The copying process creates two different processes both running the same code.
- The complex task is to ensuring that one process runs the code intended for the new process while the other process continues the work of the old process.

Scheduling in POSIX

- A process makes a copy of itself by calling the fork() function.
- That function causes the operating system to create a new process (the child process) which is a nearly exact copy of the process that called fork() (the parent process).
- They both share the same code and the same data values with one exception, the return value
- of fork().
- The parent process is returned the process ID number of the child process, while the child process gets a return value of 0.
- We can therefore test the return value of fork() to determine which process is the child

```
childid = fork();
if (childid == 0) { /* must be the child */
/* do child process here */
}
```

- execv() function takes as argument the name of the file that holds the child's code and the array of arguments.
- It overlays the process with the new code and starts executing it from the main() function.
- In the absence of an error, execv() should never return.
- The code that follows the call to perror() and exit(), take care of the case where execv() fails and returns to the parent process.
- The exit() function is a C function that is used to leave a process

```
childid = fork();
if (childid == 0) { /* must be the child */
execv("mychild",childargs);
perror("execv");
exit(1);
}
```

- The wait functions not only return the child process's status, in many implementations of POSIX they make sure that the child's resources .
- The parent stuff() function performs the work of the parent function.

```
childid = fork();
if (childid == 0) { /* must be the child */
execv("mychild",childargs);
perror("execl");
exit(1);
}
else { /* is the parent */
parent_stuff(); /* execute parent functionality */
wait(&cstatus);
exit(0);
}
```

The POSIX process model

- Each POSIX process runs in its own address space and cannot directly access the data or code.

Real-time scheduling in POSIX

- POSIX supports real-time scheduling in the POSIX_PRIORITY_SCHEDULING resource.
- POSIX supports Rate-monotonic scheduling in the SCHED_FIFO scheduling policy.
- It is a strict priority-based scheduling scheme in which a process runs until it is preempted or terminates.
- The term FIFO simply refers→ processes run in first-come first-served order.

POSIX semaphores

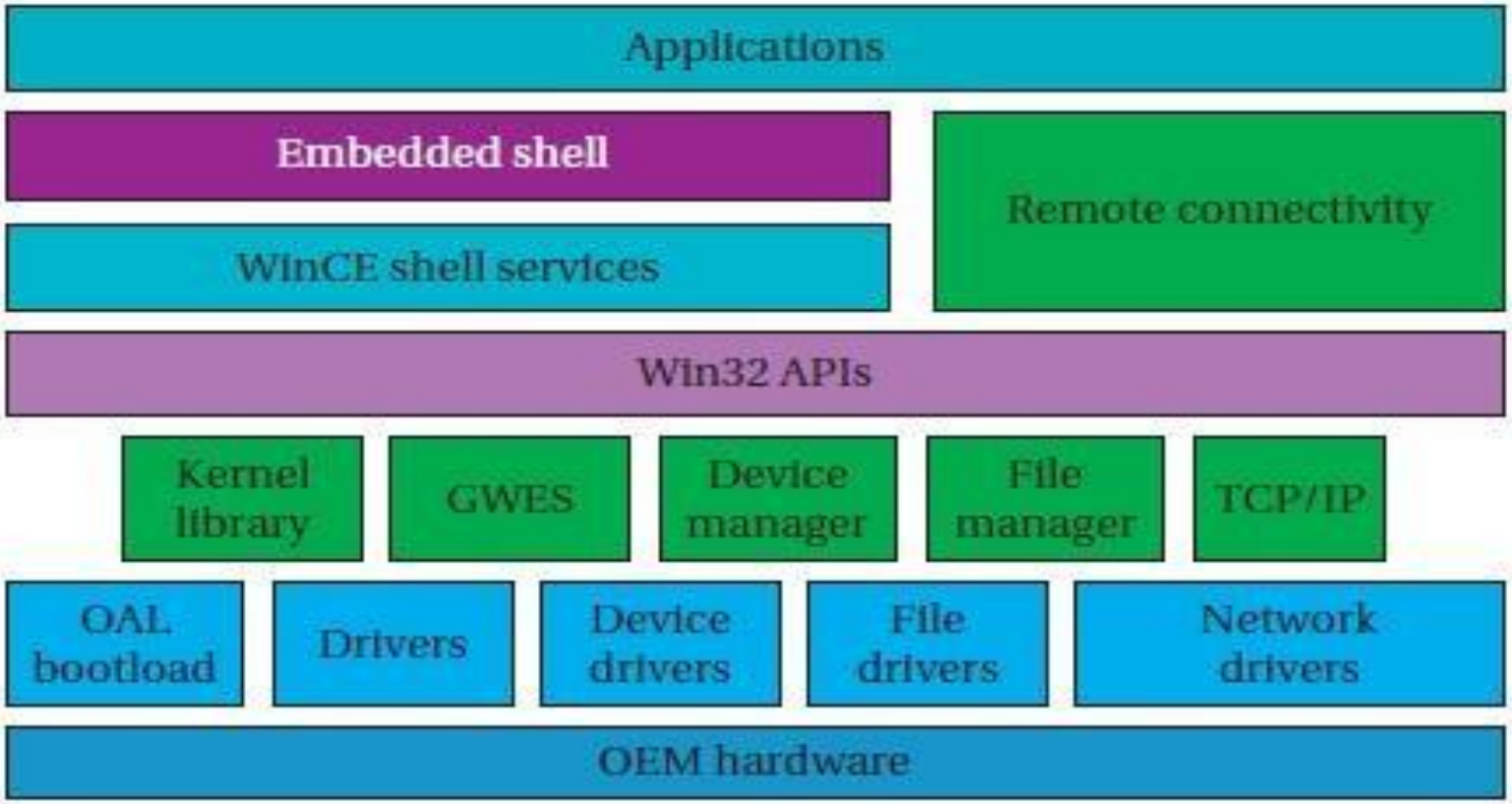
- POSIX supports semaphores and also supports a direct shared memory mechanism.
- POSIX supports counting semaphores in the _POSIX_SEMAPHORES option.
- A counting semaphore allows more than one process access to a resource at a time.
- If the semaphore allows up to N resources, then it will not block until N processes have
- simultaneously passed the semaphore;
- The blocked process can resume only after one of the processes has given up its semaphore.
- When the semaphore value is 0, the process must wait until another process gives up the semaphore and increments the count.

POSIX pipes

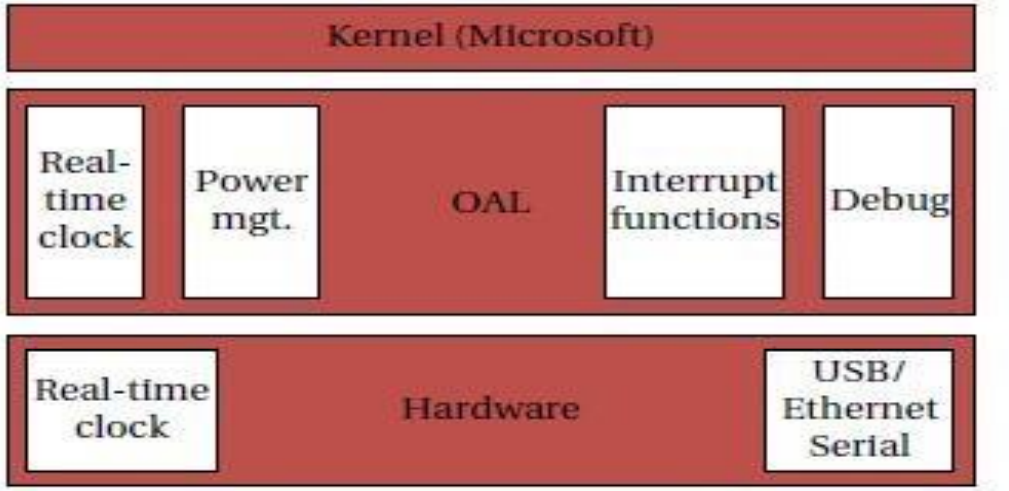
- Parent process uses the pipe() function to create a pipe to talk to a child.
- Each end of a pipe appears to the programs as a file.
- The pipe() function returns an array of file descriptors, the first for the write end and the second for the read end.
- POSIX also supports message queues under the _POSIX_MESSAGE_PASSING facility..

Windows CE

- Windows CE is designed to run on multiple hardware platforms and instruction set architectures.
- It supports devices such as smart phones, electronic instruments etc..,



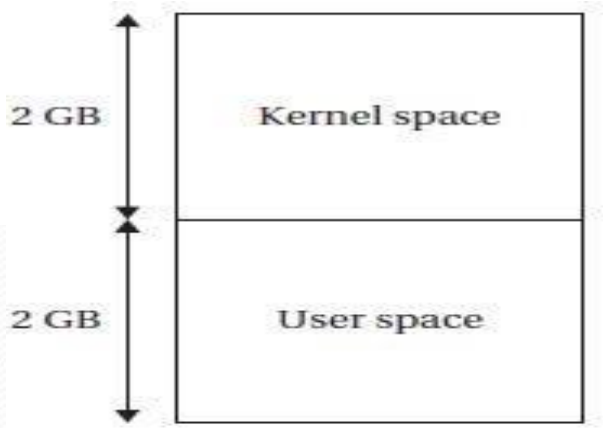
- Applications run under the shell and its user interface.
- The Win32 APIs manage access to the operating system.
- OEM Adaption Layer (OAL)→ provides an interface to the hardware and software architecture.



- OAL → provides services such as a real-time clock, power management, interrupts, and a debugging interface.
- A Board Support Package (BSP) for a particular hardware platform includes the OAL and drivers.

Memory Space

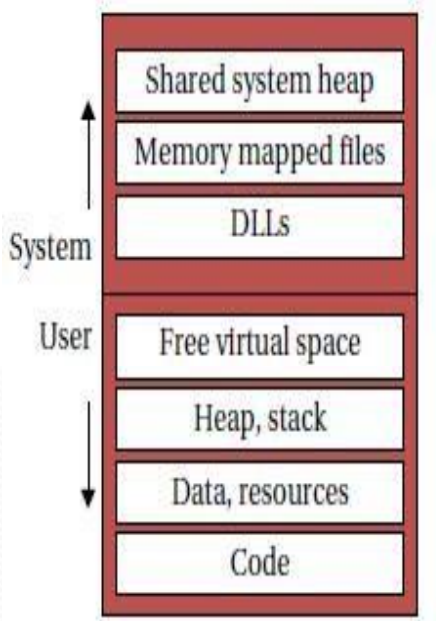
- It support for virtual memory with a flat 32-bit virtual address space.
- A virtual address can be statically mapped into main memory for key kernel-mode code.
- An address can also be dynamically mapped, which is used for all user-mode and some kernel-mode code.
- Flash as well as magnetic disk can be used as a backing store



- The top 1 GB is reserved for system elements such as DLLs, memory mapped files, and shared system heap.
- The bottom 1 GB holds user elements such as code, data, stack, and heap.

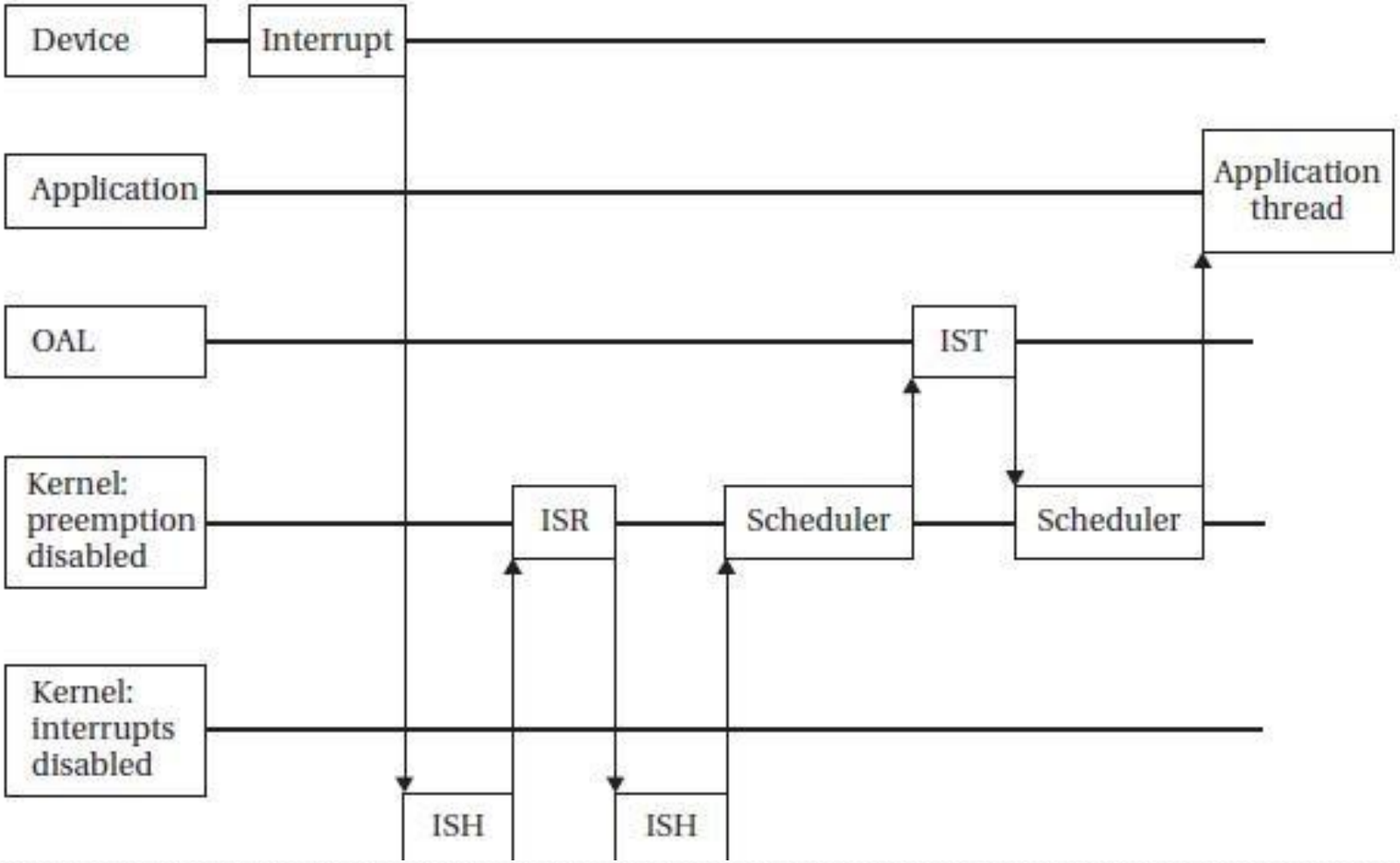
User address space in windows CE

- Threads are defined by executable files while drivers are defined by dynamically-linked libraries (DLLs).
- A process can run multiple threads.
- Threads in different processes run in different execution environments.
- Threads are scheduled directly by the operating system.
- Threads may be launched by a process or a device driver.
- A driver may be loaded into the operating system or a process.
- Drivers can create threads to handle interrupts
- Each thread is assigned an integer priority.
- 0 is the highest priority and 255 is the lowest priority.
- Priorities 248 through 255 are used for non-real-time threads .
- The operating system maintains a queue of ready processes at each priority level.



- Execution of a thread can also be blocked by a higher-priority thread.
- Tasks may be scheduled using either of two policies: a thread runs until the end of its quantum; or a thread runs until a higher-priority thread is ready to run.
- Within each priority level, round-robin scheduling is used.
- WinCE supports priority inheritance.
- When priorities become inverted, the kernel temporarily boosts the priority of the lower-priority thread to ensure that it can complete and release its resources.
- Kernel will apply priority inheritance to only one level.
- If a thread that suffers from priority inversion in turn causes priority inversion for another thread, the kernel will not apply priority inheritance to solve the nested priority inversion.

Sequence diagram for an interrupt



- Interrupt handling is divided among three entities
- The interrupt service handler (ISH)→ is a kernel service that provides the first response to the interrupt.
- The ISH selects an interrupt service routine (ISR) to handle the interrupt.
- The ISR in turn calls an interrupt service thread (IST) which performs most of the work required to handle the interrupt.
- The IST runs in the OAL and so can be interrupted by a higher-priority interrupt.
- ISR→determines which IST to use to handle the interrupt and requests the kernel to schedule that thread.
- The ISH then performs its work and signals the application about the updated device status as appropriate.
- kernel-mode and user-mode drivers use the same API.

Distributed Embedded Systems (DES)

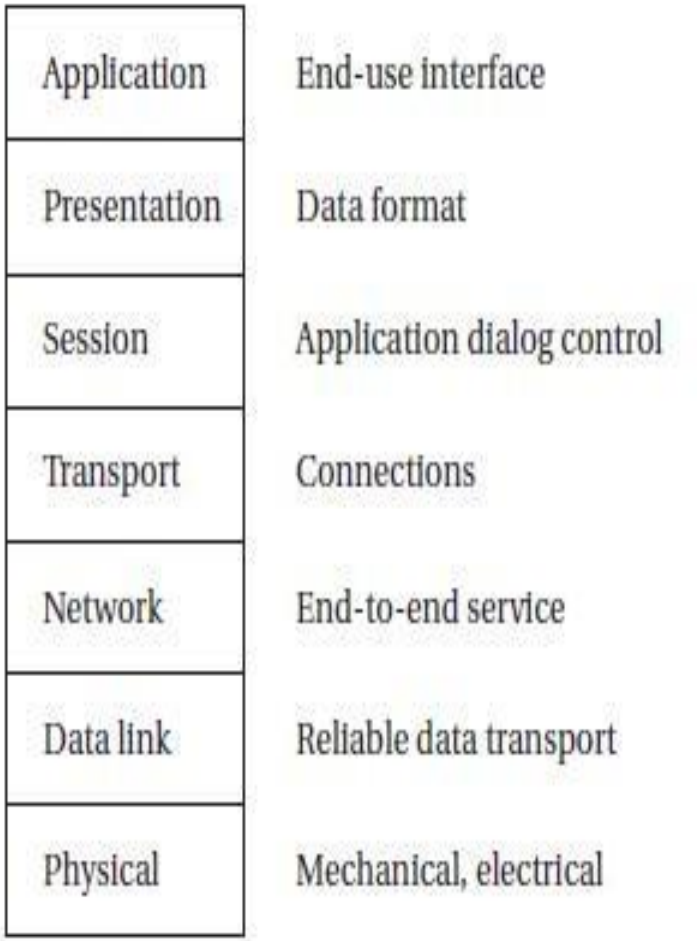
- It is a collection of hardware and software and its communication.
- It also has many control system performance.
- Processing Element (PE)is a basic unit of DES.
- It allows the network to communicate.
- PE is an instruction set processor such as DSP,CPU and Microcontroller.

Network abstractions

- Networks are complex systems.
- It provide high-level services such as data transmission from the other components in the system.
- ISO has developed a seven-layer model for networks known as Open Systems Interconnection (OSI) models.

OSI model layers

- Physical layer→ defines the basic properties of the interface between systems, including the physical connections, electrical properties & basic procedures for exchanging bits.
- Data link layer→ used for error detection and control across a single link.
- Network layer→ defines the basic end-to-end data transmission service.
- Transport layer→ defines connection-oriented services that ensure that data are delivered in the proper order .
- Session layer→ provides mechanisms for controlling the interaction of end-user services across a network, such as data grouping and checkpointing.
- Presentation layer→ layer defines data exchange formats
- Application layer→ provides the application interface between the network and end-user programs.

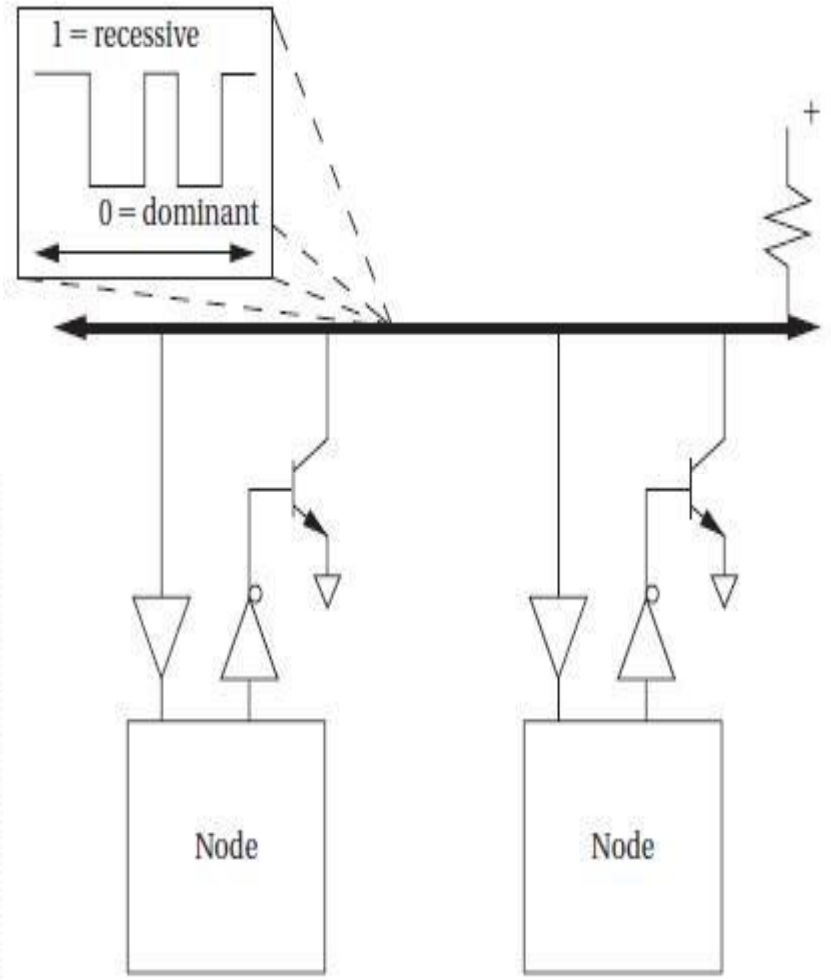


Controller Area Network(CAN)Bus

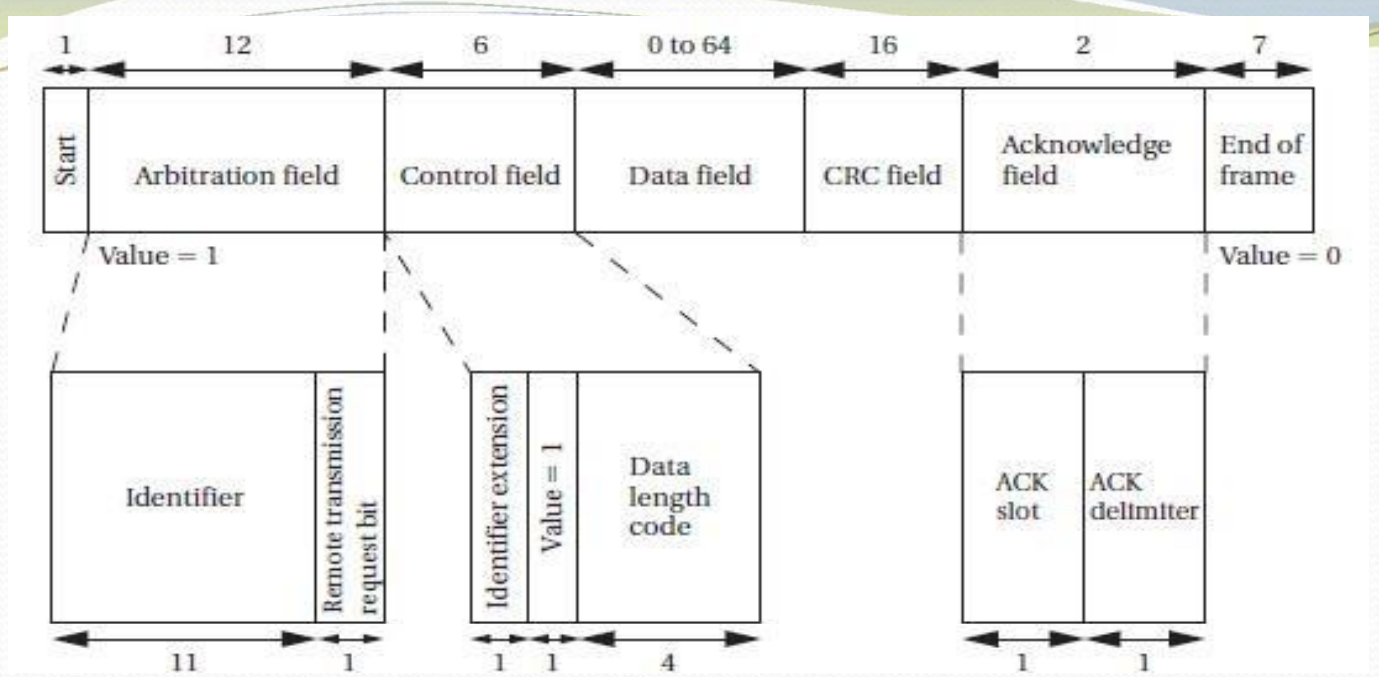
- It was designed for automotive electronics and was first used in production cars in 1991.
- It uses bit-serial transmission.
- CAN can run at rates of 1 Mbps over a twisted pair connection of 40 meters.
- An optical link can also be used.

4.7.2.1)Physical-electrical organization of a CAN bus

- Each node in the CAN bus has its own electrical drivers and receivers that connect the node to the bus in wired-AND fashion.
- When all nodes are transmitting 1s, the bus is said to be in the recessive state.
- when a node transmits a 0s, the bus is in the dominant state.



Data Frame



- Arbitration field→ The first field in the packet contains the packet's destination address 11 bits
- Remote Transmission Request (RTR) bit is set to 0 if the data frame is used to request data from the destination identifier.
- When RTR = 1, the packet is used to write data to the destination identifier.
- Control field→ 4-bit length for the data field with a 1 in between.
- Data field→0 to 64 bytes, depending on the value given in the control field.
- CRC→ It is sent after the data field for error detection.
- Acknowledge field →identifier signal whether the frame was correctly received.(sender puts a bit (1) in the ACK slot , if the receiver detected an error, it put (0) value)

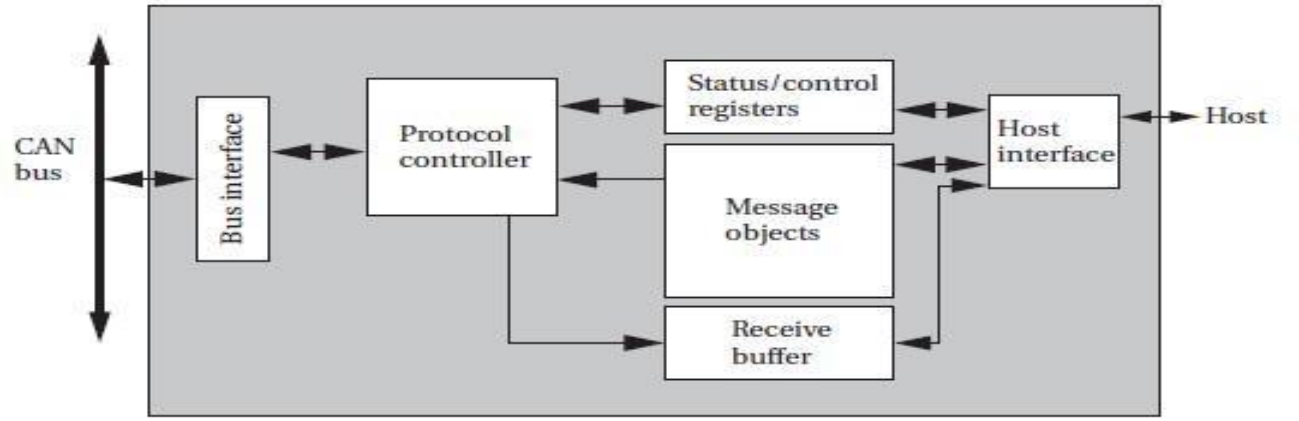
Arbitration

- It uses a technique known as Carrier Sense Multiple Access with Arbitration on Message Priority (CSMA/AMP).
- When a node hears a dominant bit in the identifier when it tries to send a recessive bit, it stops transmitting.
- By the end of the arbitration field, only one transmitter will be left.
- The identifier field acts as a priority identifier, with the all-o having the highest priority

Error handling

- An error frame can be generated by any node that detects an error on the bus.
- Upon detecting an error, a node interrupts the current transmission.
- Error flag field followed by an error delimiter field of 8 recessive bits.
- Error delimiter field allows the bus to return to the quiescent state so that data frame transmission can resume.
- Overload frame signals that a node is overloaded and will not be able to handle the next message. Hence the node can delay the transmission of the next frame .

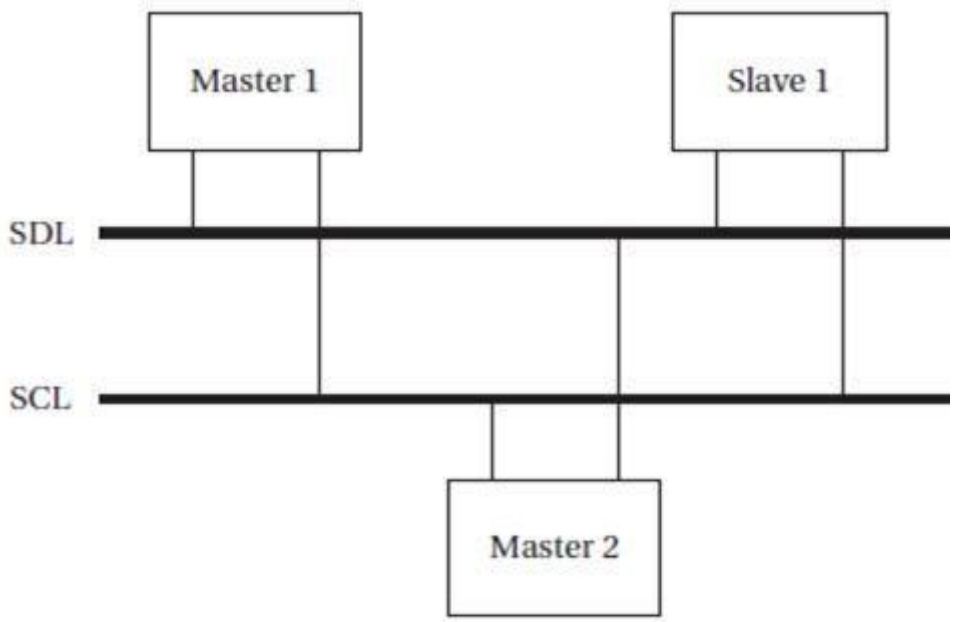
Architecture of a CAN controller



- The controller implements the physical and data link layers.
- CAN does not need network layer services to establish end-to-end connections.
- The protocol control block is responsible for determining when to send messages, when a message must be resent and when a message should be received.

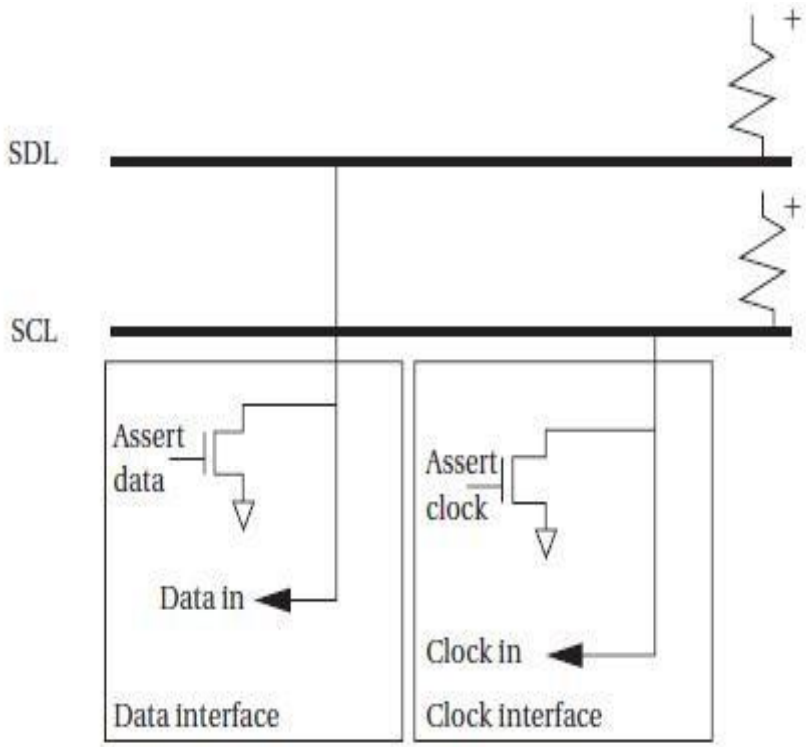
I²C bus

- I²C bus→ used to link microcontrollers into systems.
- I²C is designed to be low cost, easy to implement, and of moderate speed (up to 100kbps for the standard bus and up to 400 kbps for the extended bus).
- Serial data line (SDL) for data transmission.
- Serial clock line (SCL)→ indicates when valid data are on the data line.
- Every node in the network is connected to both SCL and SDL.
- Some nodes may act as bus masters .
- Other nodes may act as slaves that only respond to requests from masters.



Electrical interface to the I2C bus

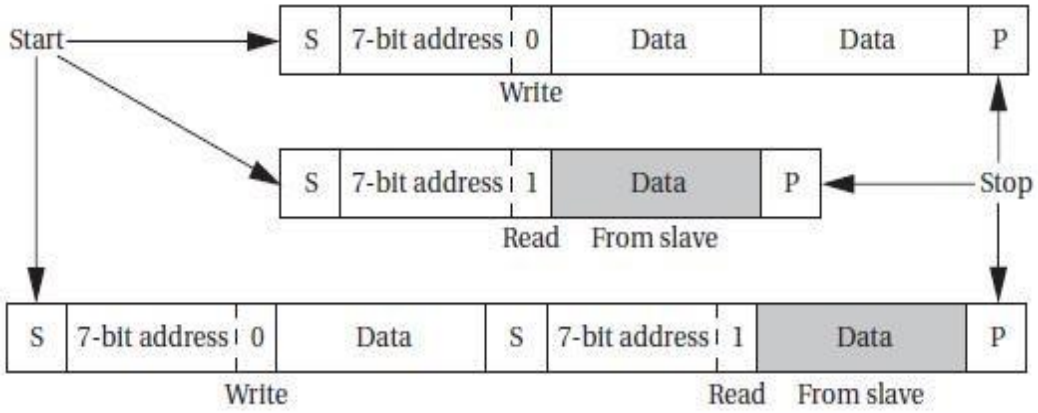
- Both bus lines are defined by an electrical signal.
- Both bus signals use open collector/open drain circuits.
- The open collector/open drain circuitry allows a slave device to stretch a clock signal during a read.
- The master is responsible for generating the SCL clock.
- The slave can stretch the low period of the clock.
- It is a multi master bus so different devices may act as the master at various times.
- Master drives both SCL and SDL when it is sending data.
- When the bus is idle, both SCL and SDL remain high.
- When two devices try to drive either SCL or SDL , the open collector/open drain circuitry prevents errors.
- Each master device make sure that it is not interfering with another message.



Format of an I2C address transmission

- Every I²C device has an separate address.
- A device address is 7 bits and 1 bit for read/write data.
- The address 0000000 ,which can be used to signal all devices simultaneously.
- The address 1110XX is reserved for the extended 10-bit addressing scheme.

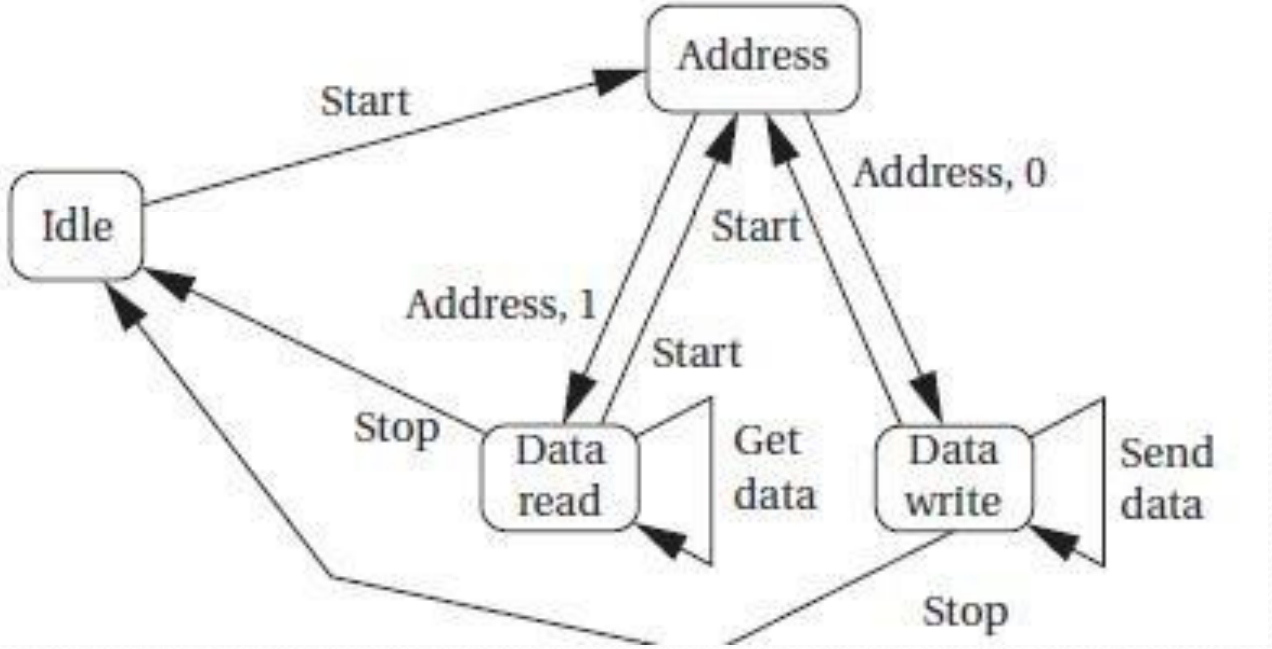
Bus transactions on the I2C bus



- When a master wants to write a slave, it transmits the slave's address followed by the data.
- When a master send a read request with the slave's address and the slave transmit the data.
- Transmission address has 7-bit and 1 bit for data direction.(0 for writing from the master to the slave and 1 for reading from the slave to the master)
- A bus transaction is initiated by a start signal and completed with an end signal.
- A start is signaled by leaving the SCL high and sending a 1 to 0 transition on SDL.
- A stop is signaled by setting the SCL high and sending a 0 to 1 transition on SDL.

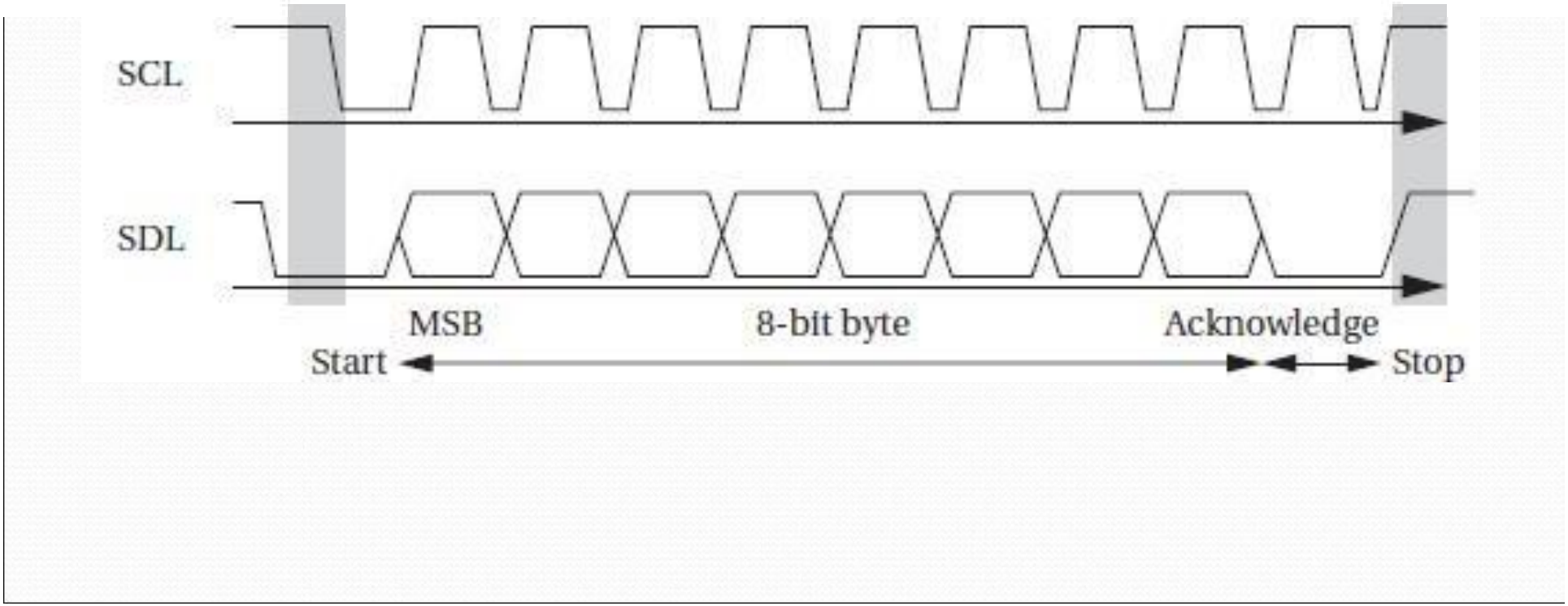
State transition graph for an I2C bus master

- Starts and stops must be paired.
- A master can write and then read by sending a start after the data transmission, followed by another address transmission and then more data.



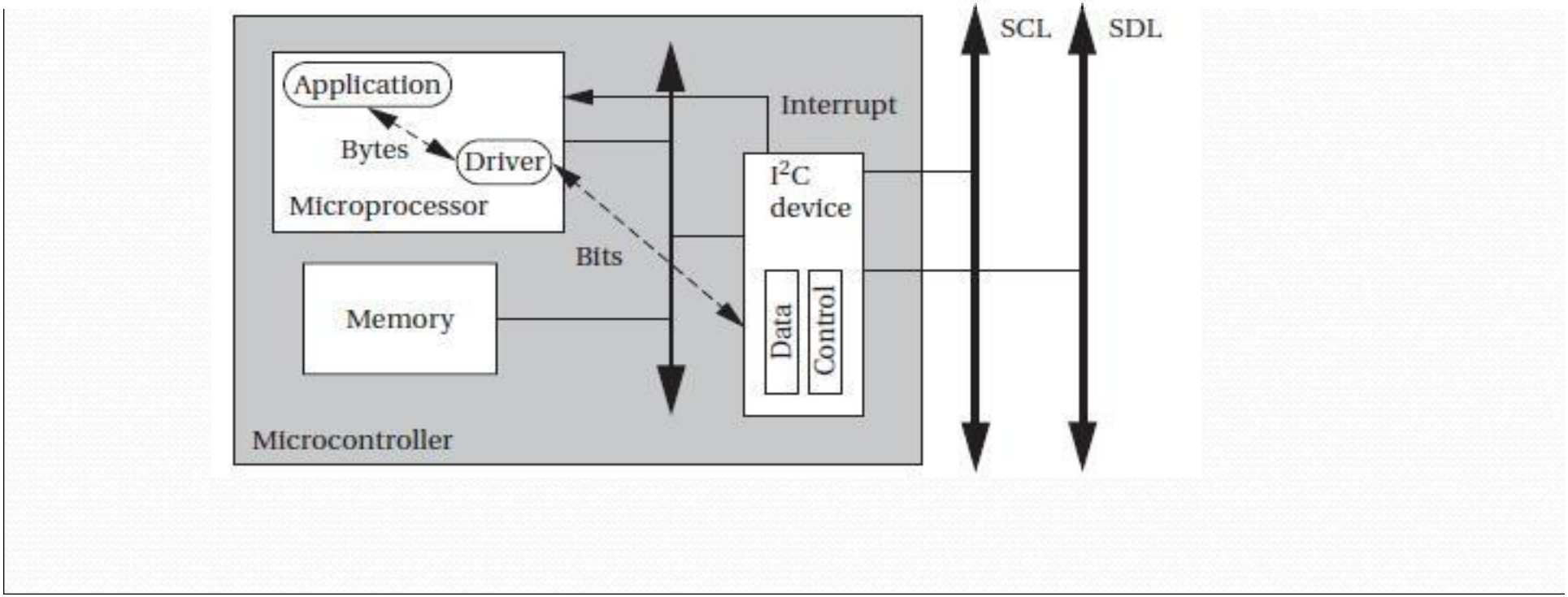
Transmitting a byte on the I2C bus

- The transmission starts when SDL is pulled low while SCL remains high.
- The clock is pulled low to initiate the data transfer.
- At each bit, the clock goes high while the data line assumes its proper value of 0 or 1.
- An acknowledgment is sent at the end of every 8-bit transmission, whether it is an address or data.
- After acknowledgment, the SDL goes from low to high while the SCL is high, signaling the stop condition.



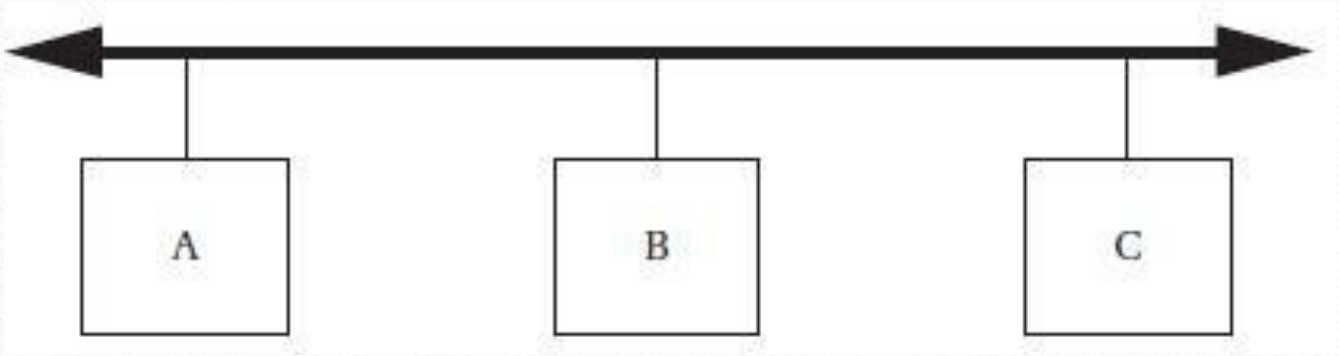
I2C interface in a microcontroller

- System has a 1-bit hardware interface with routines for byte-level functions.
- I²C device used to generates the clock and data.
- Application code calls routines to send an address, data byte, and also generates the SCL ,SDL and acknowledges.
- Timers is used to control the length of bits on the bus.
- When Interrupts used in master mode, polled I/O may be acceptable.
- If no other pending tasks can be performed, because masters initiate their own transfers.



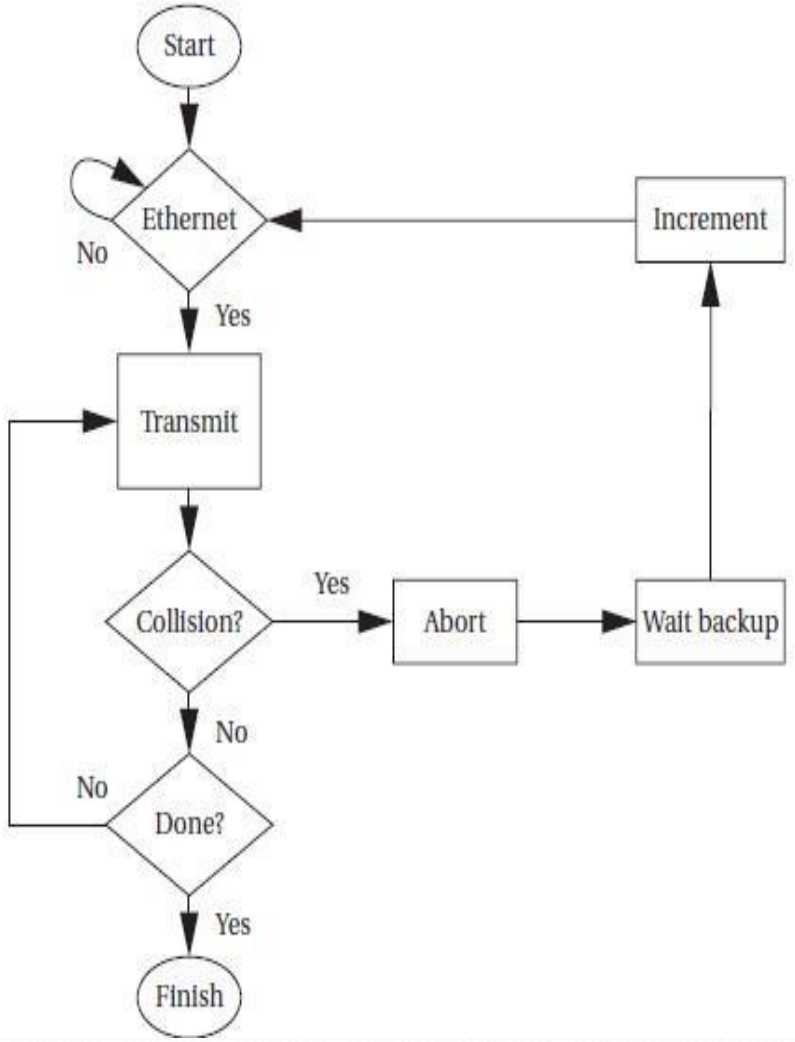
ETHERNET

- It is widely used as a local area network for general-purpose computing.
- It is also used as a network for embedded computing.
- It is particularly useful when PCs are used as platforms, making it possible to use standard components, and when the network does not have to meet real-time requirements.
- It is a bus with a single signal path.
- It supports both twisted pair and coaxial cable.
- Ethernet nodes are not synchronized, if two nodes decide to transmit at the same time,the message will be ruined.

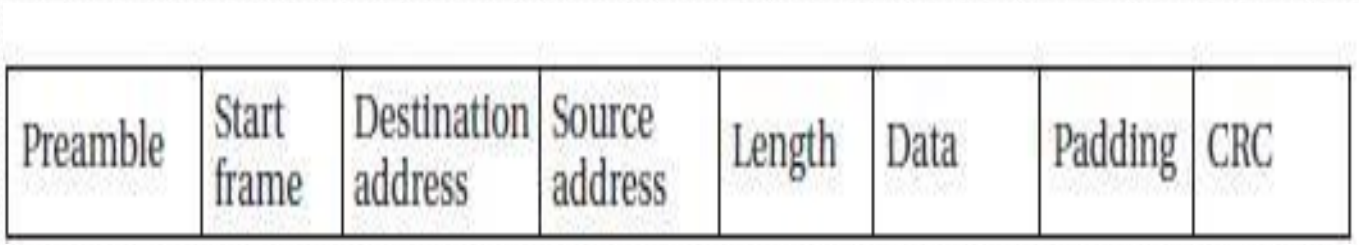


Ethernet CSMA/CD algorithm

- A node that has a message waits for the bus to become silent and then starts transmitting.
- It simultaneously listens, and if it hears another transmission that interferes with its transmission, it stops transmitting and waits to retransmit.
- The waiting time is random, but weighted by an exponential function of the number of times the message has been aborted



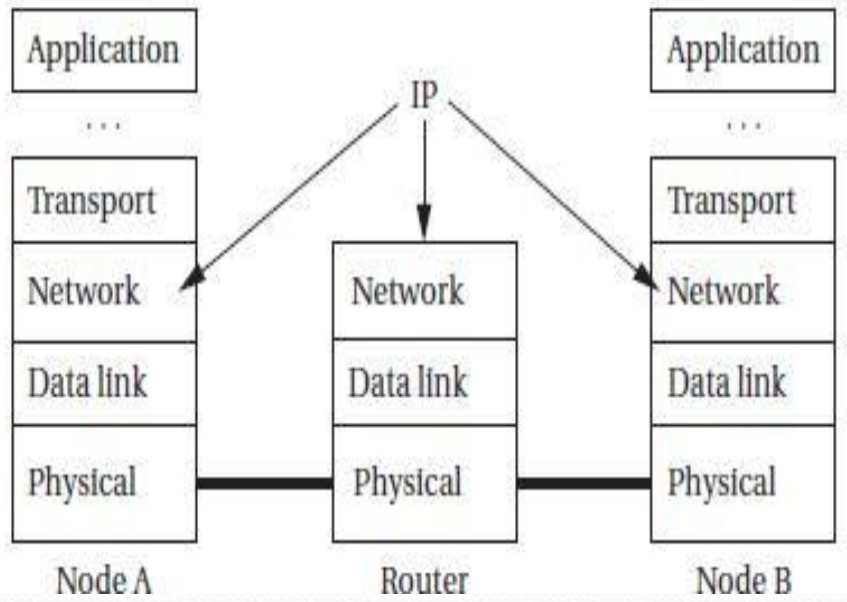
Ethernet-Packet format



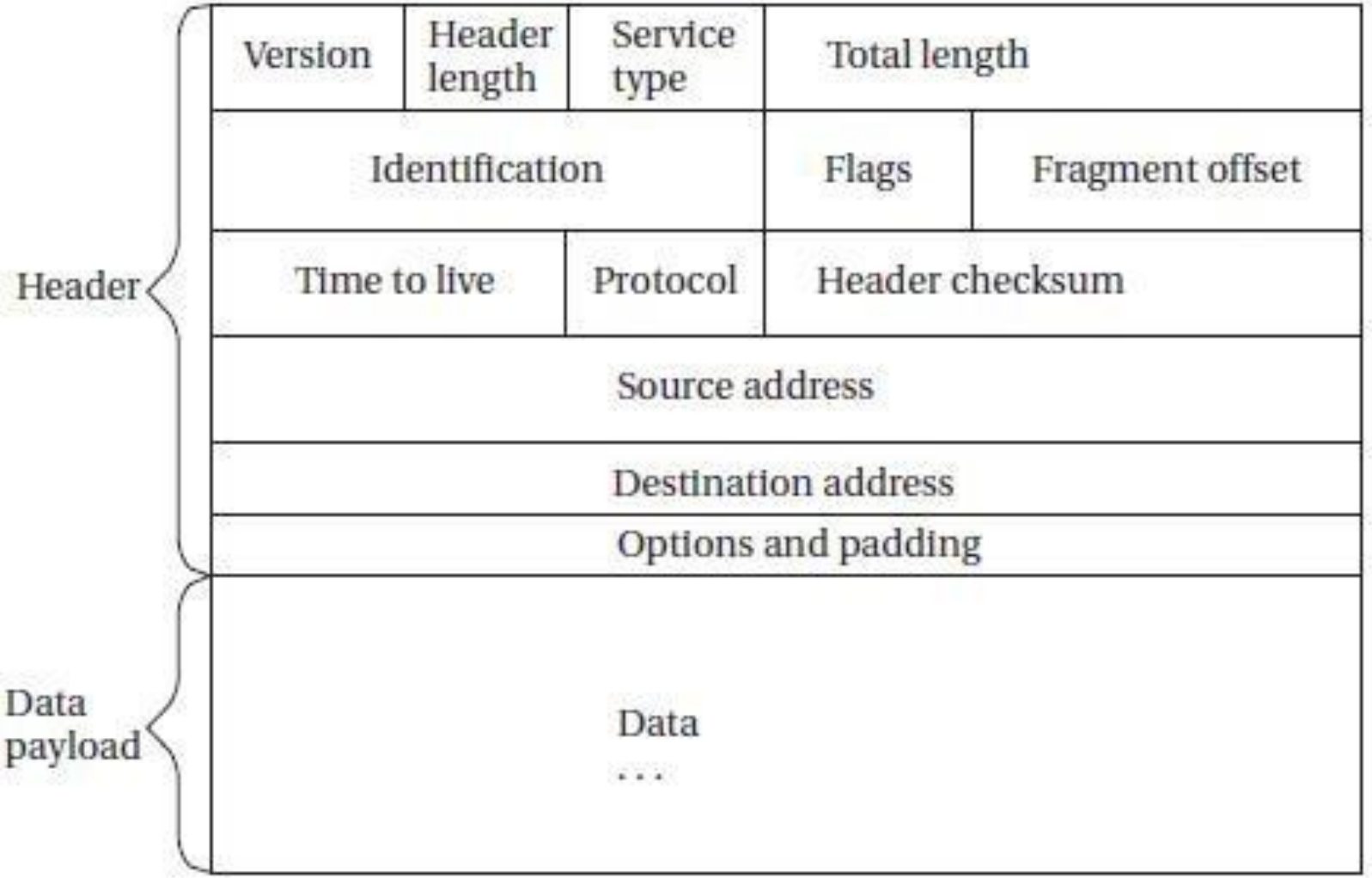
- Preamble→ 56-bit of alternating 1 and 0 bits, allowing devices on the network to easily synchronize their receiver clocks.
- SFD→8-bit ,indicates the beginning of the Ethernet frame
- Physical or MAC addresses → destination and the source(48-bit length)
- Length data payload→The minimum payload is 42 octets

INTERNET PROTOCOL(IP)

- It is the fundamental protocol on the Internet.
- It provides connection orientded, packet-based communication.
- It transmits packet over different networks from source to destination.
- It allows data to flow seamlessly from one end user t another.
- When node A wants to send data to node B, the data pass through several layers of the protocol stack to get to the Internet Protocol.
- IP creates packets for routing to the destination, which are then sent to the data link and physical layers.
- A packet may go through many routers to get to its destination.
- IP works at the network layer → does not guarantee that a packet is delivered to its destination.
- It supports best-effort routing packets→ packets that do arrive may come out of order.



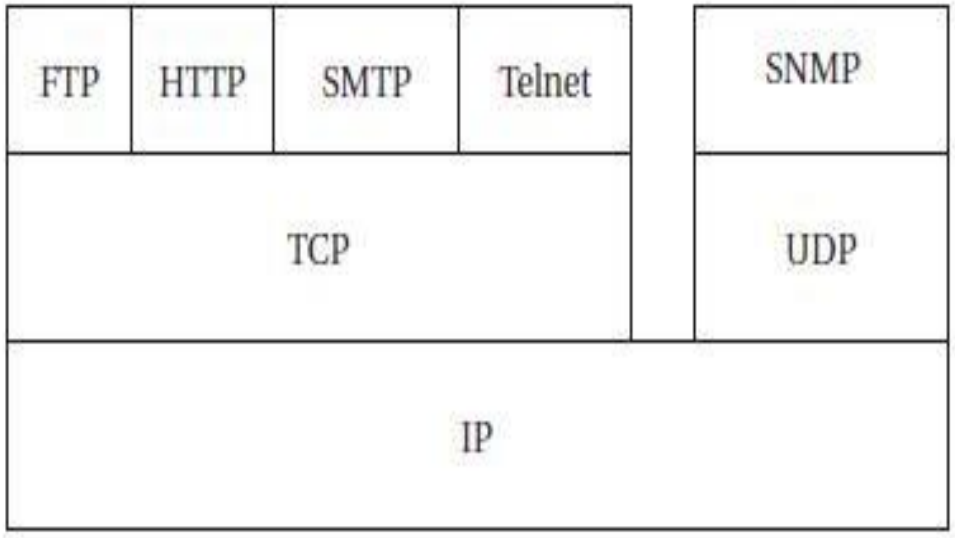
IP packet structure



- Version→ it ia s 4-bit field.used to identify v4 or v6.
- Header Length (HL)→It is a 4 bits, field.Indicates the length of the header.
- Service Type→it is a 8 bit field ,used to specify the type of service.
- Total length→Including header and data payload is 65,535 bytes.
- Identification→ identifying the group of fragments of a single IP datagram.
- Flags→ bit 0 Reserved.
  - bit 1: Don't Fragment (DF)
  - bit 2: More Fragments (MF)
- Fragment Offset→ It is 13 bits long , specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram
- Time To Live (TTL)→It is a 8 bit wide, indicates th datagram's lifetime
- Protocol→ protocol used in the data portion of the IP datagram
- Header Checksum→(16 bit) used for error-checking of the header
- Source address→ Sender packet address(32-bits size)
- Destination address→ Receiver packet address(32-bits size)

Transmission Control Protocol(TCP)

- It provides a connection-oriented service.
- It ensures that data arrive in the appropriate order.
- It uses an acknowledgment protocol to ensure that packets arrive.
- TCP is used to provide File Transport Protocol (FTP) for batch file transfers.
- Hypertext Transport Protocol (HTTP) for World Wide Web service.
- Simple Mail Transfer Protocol (SMTP) for email.
- Telnet for virtual terminals.
- User Datagram Protocol (UDP), is used to provide connection-less services.
- Simple Network Management Protocol (SNMP) provides the network management services.



MPSoCs and shared memory multiprocessors

- Shared memory processors are well-suited to applications that require a large amount of data to be processed(Signal processing systems)
- Most MPSoCs are shared memory systems.
- Shared memory allows for processors to communicate with varying patterns.
- If the pattern of communication is very fixed and if the processing of different steps is performed in different units, then a networked multiprocessor may be most appropriate.
- If one processing element is used for several different steps, then shared memory also allows the required flexibility in communication.

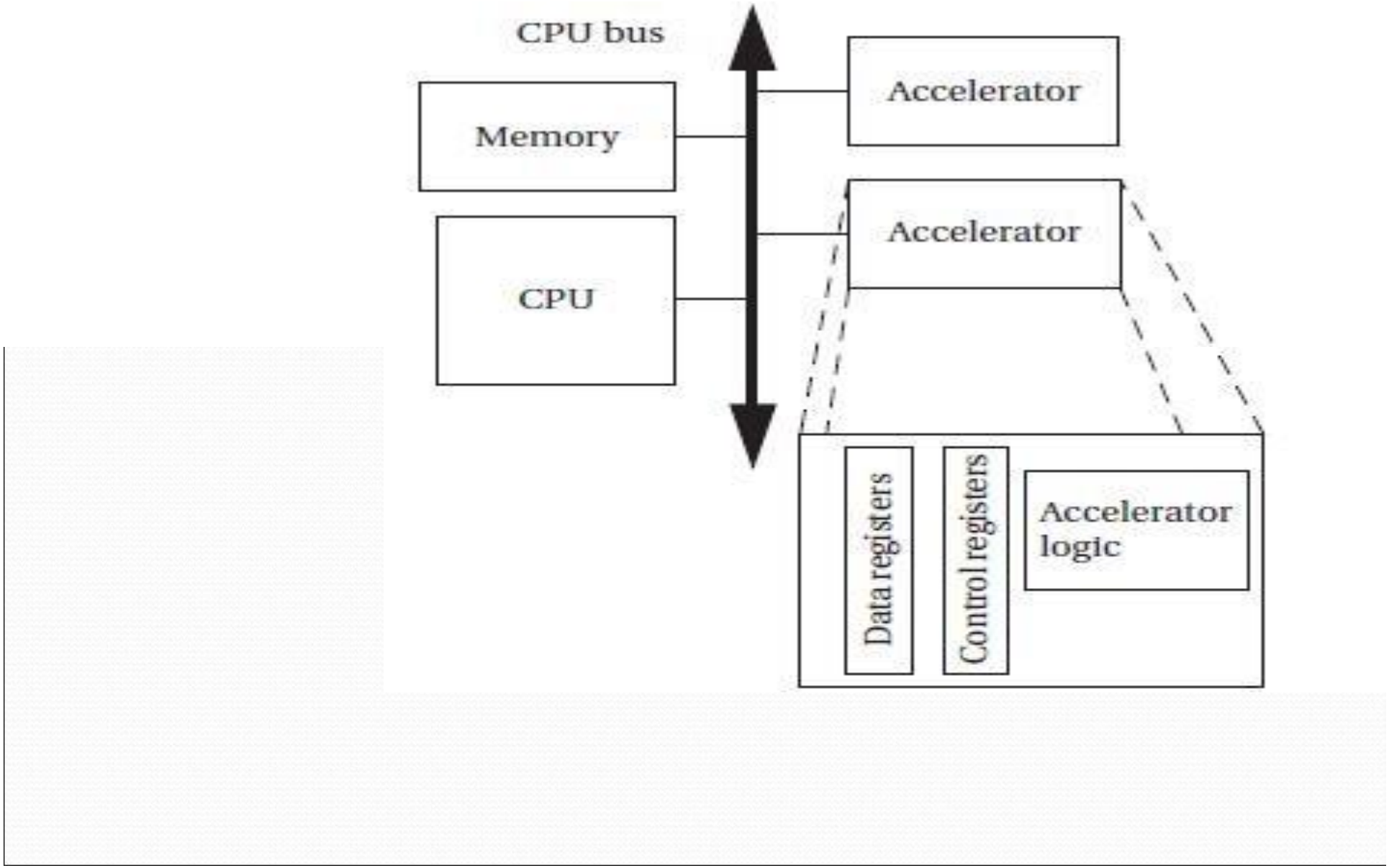
Heterogeneous shared memory multiprocessors

- Many high-performance embedded platforms are heterogeneous multiprocessors.
- Different processing elements (PE)perform different functions.
- PEs may be programmable processors with different instruction sets or specialized accelerators.
- Processors with different instruction sets can perform different tasks faster and using less energy.
- Accelerators provide even faster and lower-power operation for a narrow range of functions.

Accelerators

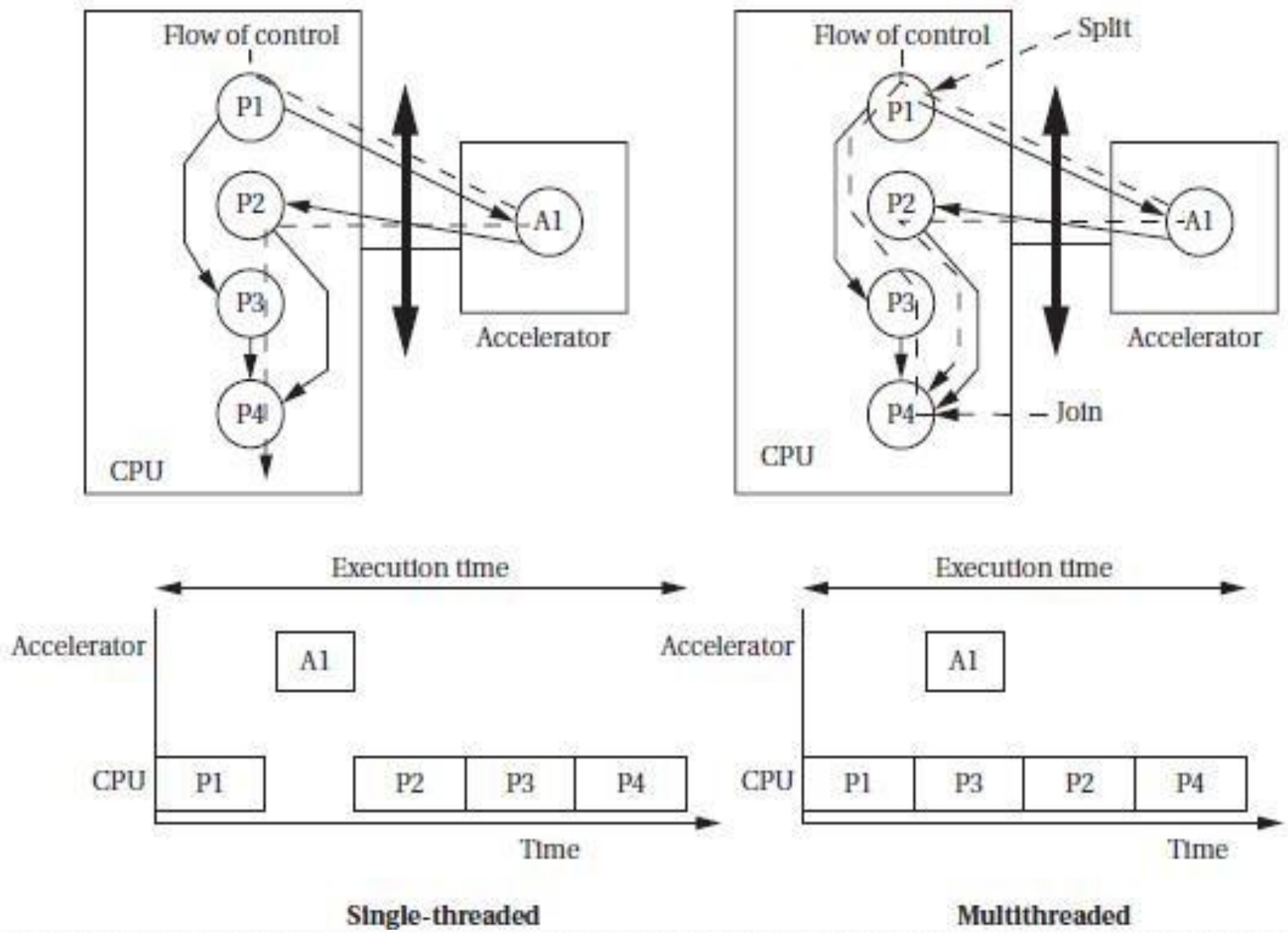
- It is the important processing element for embedded multiprocessors.
- It can provide large performance increases for applications with computational kernels .
- It can also provide critical speedups for low-latency I/O functions.
- CPU(host) accelerator is attached to the CPU bus.
- CPU talks to the accelerator through data and control registers in the accelerator.
- Control registers allow the CPU to monitor the accelerator's operation and to give the accelerator commands.
- The CPU and accelerator may also communicate via shared memory.
- The accelerator operate on a large volume of data with efficient data in memory.
- Accelerator read and write memory directly .
- The CPU and accelerator use synchronization mechanisms to ensure that they do not
- destroy each other's data.
- An accelerator is not a co-processor.
- A co-processor is connected to the internals of the CPU and processes instructions.
- An accelerator interacts with the CPU through the programming model interface.
- It does not execute instructions.
- CPU and accelerators performs computations for specification.

CPU accelerators in a system



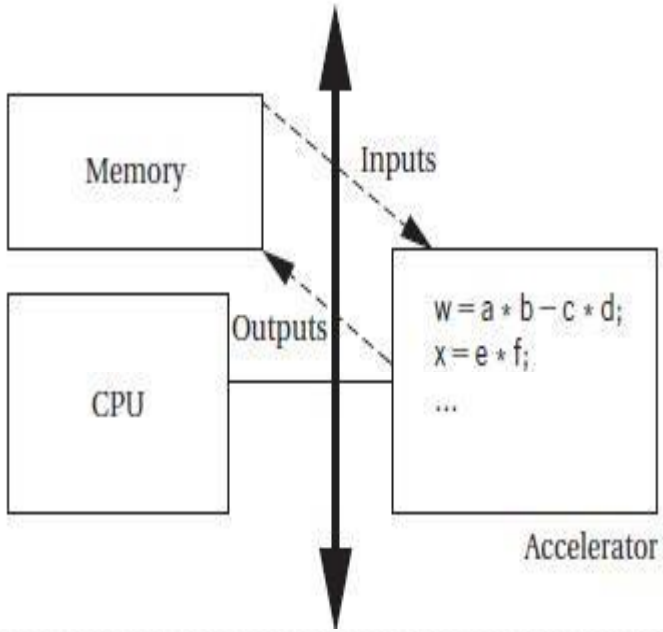
Accelerator Performance Analysis

- The speed factor of accelerator will depend on the following factors.
- Single threaded→CPU is in idle state while the accelerator runs.
- Multithreaded→CPU do some useful work in parallel with accelerator.
- Blocking→ CPU's scheduler block other operations wait for the accelerator call to complete.
- Non-blocking→ CPU's run some other work parallel with accelerator.
- Data dependencies allow P2 and P3 to run independently on the CPU.
- P2 relies on the results of the A1 process that is implemented by the accelerator.
- Single-threaded→ CPU blocks to wait for the accelerator to return the results of its computation.t, it doesn't matter whether P2 or P3 runs next on the CPU.
- Multithreaded → CPU continues to do useful work while the accelerator runs, so the CPU can start P3 just after starting the accelerator and finish the task earlier.


Flow of control
P1
P2
P3
P4
A1
Accelerator
CPU
Flow of control
Split
P1
P2
P3
P4
A1
Accelerator
Join
CPU
Execution time
Accelerator
A1
CPU
P1
P2
P3
P4
Time
Single-threaded
Execution time
Accelerator
A1
CPU
P1
P3
P2
P4
Time
Multithreaded

Components of execution time for an accelerator

- Execution time of a accelerator depends on the time required to execute the accelerator's function.
- It also depends on the time required to get the data into the accelerator and back out of it.
- Accelerator will read all its input data, perform the required computation,and write all its results.
- Total execution time given as
- $t_{acccel}=t_x+t_{in}+t_{out}$
- t_x →execution time of the accelerator
- T_{in} →times required for reading the required variables
- t_{out}- →times required for writing the required variables

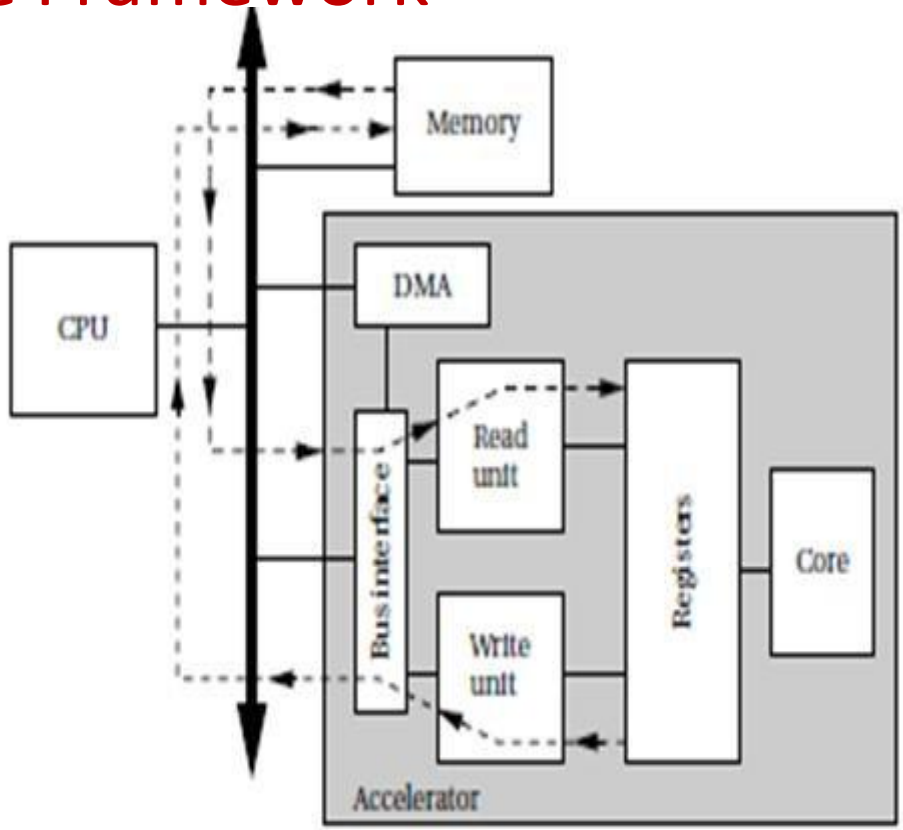


System Architecture Framework

- Architectural design depends on the application.

An accelerator can be considered from two angles.

- Accelerator core functionality
- Accelerator interface to the CPU bus.
- The accelerator core typically operates off internal registers.
- Requirement of number of registers is an important design decision.
- Main memory accesses will probably take multiple clock cycles.
- Status registers used to test the accelerator's state an to perform basic operations(starting, stopping, and resetting the accelerator)
- A register file in the accelerator acts as a buffer betwe main memory and the accelerator core.
- Read unit can read the accelerator's requirements and load the registers with the next required data.
- Write unit can send recently completed values to main memory.

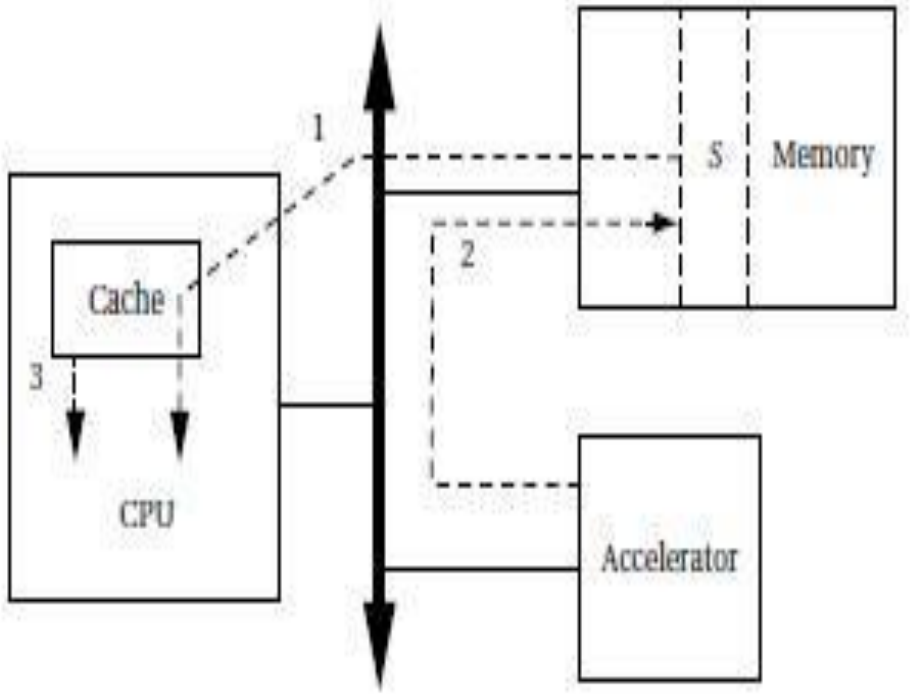


cache problem in an accelerated system

- CPU cache can cause problems for accelerators.

1. The CPU reads location S.
2. The accelerator writes S.
3. The CPU again reads S.

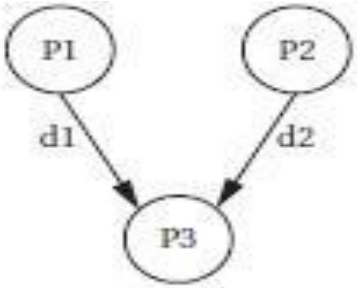
- If the CPU has cached location S ,the program will not see the value of S written by the accelerator. It will instead get the old value of S stored in the cache
- To avoid this problem, the CPU's cache must update the cache by setting cache entry is invalid.



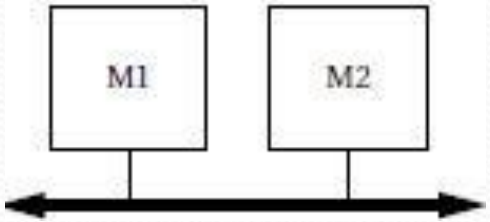
Scheduling and allocation

- Designing a distributed embedded system, depends upon the scheduling and allocation of resources.
- We must schedule operations in time, including communication on the network and computations on the processing elements.
- The scheduling of operations on the PEs and the communications between the PEs are linked.
- If one PE finishes its computations too late, it may interfere with another communication on the network as it tries to send its result to the PE that needs it.
- This is bad for both the PE that needs the result and the other PEs whose communication is interfered with.
- We must allocate computations to the processing elements.
- The allocation of computations to the PEs determines what communications are required—if a value computed on one PE is needed on another PE, it must be transmitted over the network.

- We can specify the system as a task graph. However, different processes may end up on different processing elements. Here is a task graph



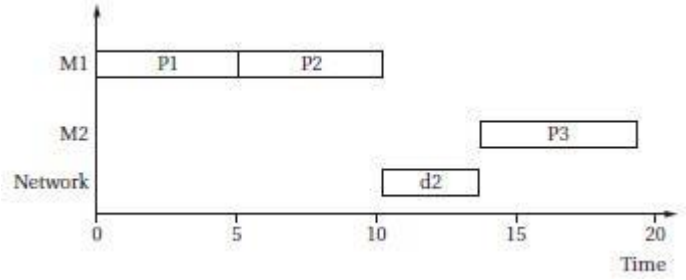
- We have labeled the data transmissions on each arc ,We want to execute the task on the platform below.



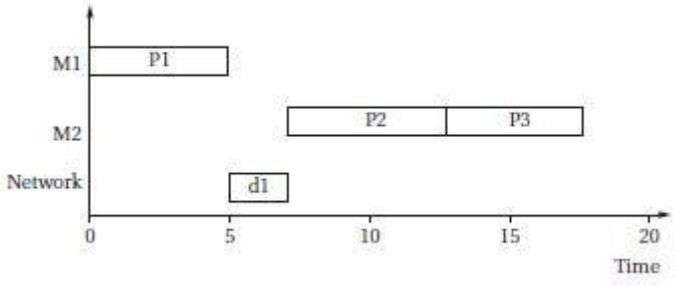
- The platform has two processing elements and a single bus connecting both PEs. Here are the process speeds:

	M1	M2
P1	5	5
P2	5	6
P3	—	5

- As an initial design, let us allocate P1 and P2 to M1 and P3 to M2This schedule shows what happens on all the processing elements and the network.



- The schedule has length 19. The d1 message is sent between the processes internal to
- P1 and does not appear on the bus.
- Let's try a different allocation. P1 on M1 and P2 and P3 on M2. This makes P2 run more slowly. Here is the new schedule:.
- The length of this schedule is 18, or one time unit less than the other schedule. The
- increased computation time of P2 is more than made up for by being able to transmit a
- shorter message on the bus. If we had not taken communication into account when analyzing total execution time, we could have made the wrong choice of which processes to put on the same processing element



Audio player/MP3 Player

Operation and requirements

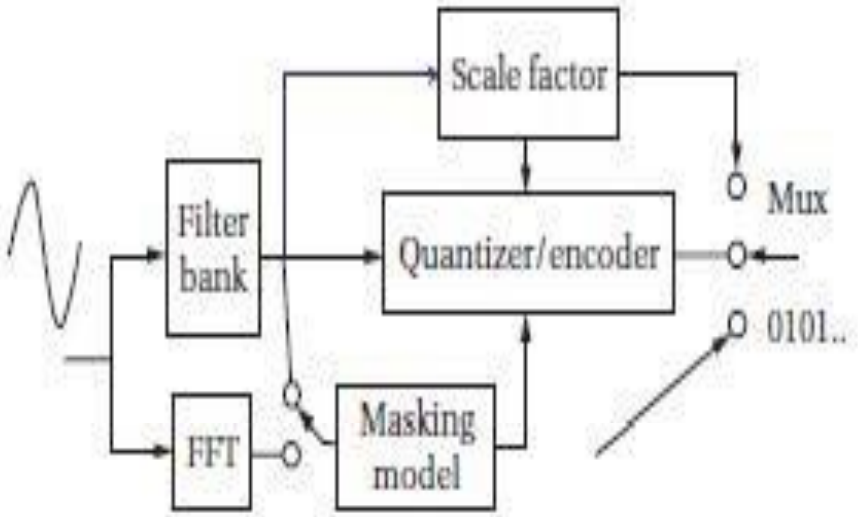
- MP3 players use either flash memory or disk drives to store music.
- It performs the following functions such as audio storage, audio decompression, and user interface.
- Audio compression→ It is a lossy process. The coder eliminates certain features of the audio stream so that the result can be encoded in fewer bits.
- Audio decompression→ The incoming bit stream has been encoded using a Huffman style code, which must be decoded.
- Masking→ One tone can be masked by another if the tones are sufficiently close in frequency.

Audio compression standards

- Layer 1 (MP1) →uses a lossless compression of sub bands and simple masking model.
- Layer 2 (MP2) →uses a more advanced masking model.
- Layer 3 (MP3)→ performs additional processing to provide lower bit rates.

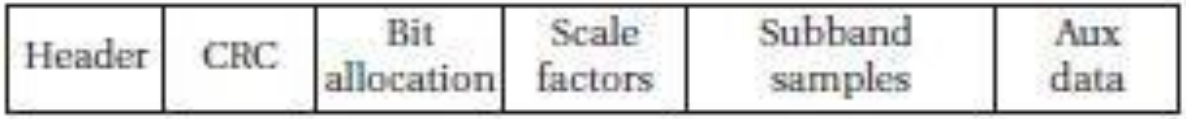
MPEG Layer 1 encoder

- Filter bank→ splits the signal into a set of 32 sub-bands that are equally spaced in the frequency domain and together cover the entire frequency range of the audio.
- Encoder→It reduce the bit rate for the audio signals.
- Quantizer →scales each sub-band(fits within 6 bits), then quantizes based upon the current scale factor for that sub-band.
- Masking model → It is driven by a separate Fast Fourier transform (FFT), the filter bank could be used for masking, a separate FFT provides better results.
- The masking model chooses the scale factors for the sub-bands, which can change along with the audio stream.
- Multiplexer→ output of the encoder passes along all the required data.



MPEG Layer 1 data frame format

- A frame carries the basic MPEG data, error correction codes, and additional information.
- After disassembling the data frame, the data are un-scaled and inverse quantized to produce sample streams for the sub-band.



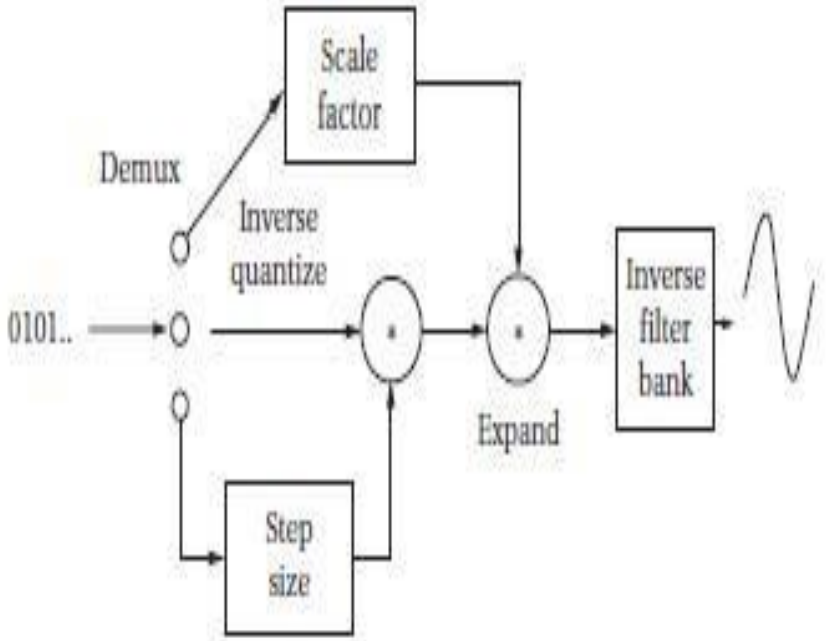
MPEG Layer 1 decoder

•After disassembling the data frame, the data are un-scaled and inverse quantized to produce sample streams for the sub-band.

•An inverse filter bank then reassembles the sub-bands into the uncompressed signal.

User interface→ MP3 player is simple both the physical size and power consumption of the device. Many players provide only a simple display and a few buttons.

File system→ player generally must be compatible with PCs. CD/MP3 players used compact discs that had been created on PCs.

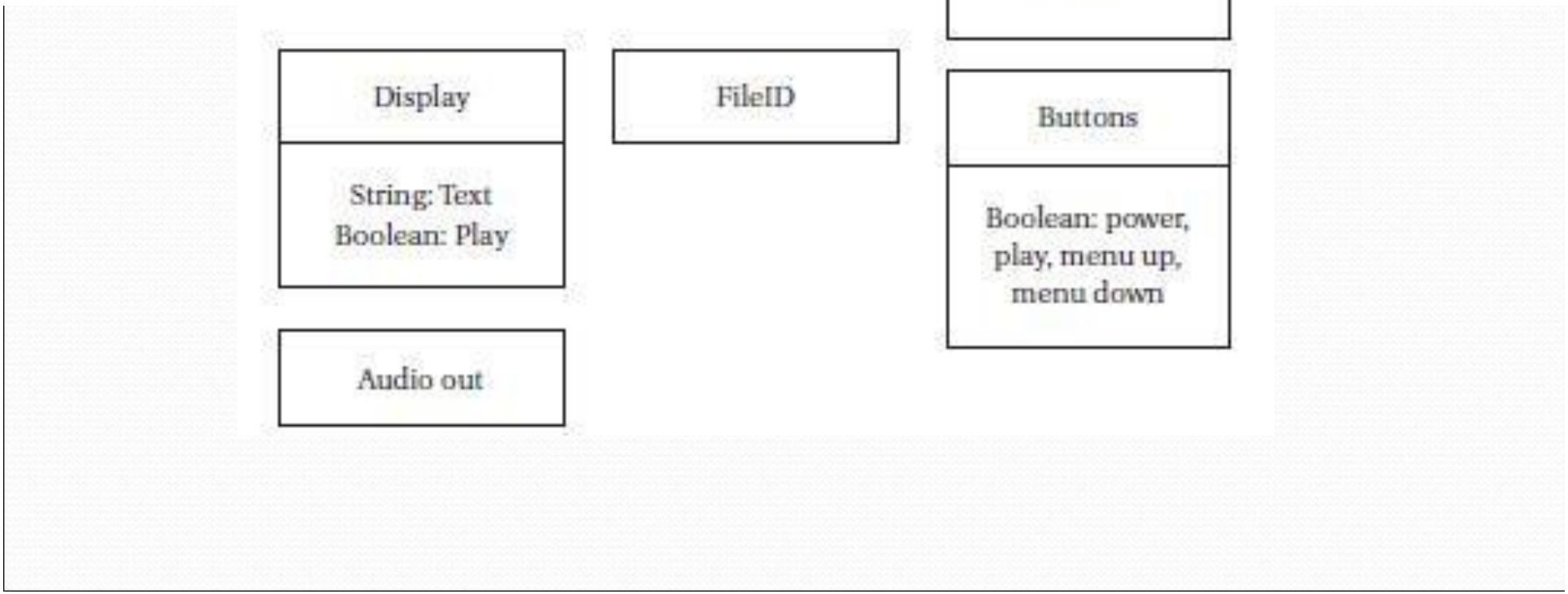


Requirements

Name	Audio player
Purpose	Play audio from files.
Inputs	Flash memory socket, on/off, play/stop, menu up/down.
Outputs	Speaker
Functions	Display list of files in flash memory, select file to play, play file.
Performance	Sufficient to play audio files at required rate.
Manufacturing cost	Approximately $25
Power	1 AAA battery
Physical size and weight	Approx. 1 in x 2 in, less than 2 oz

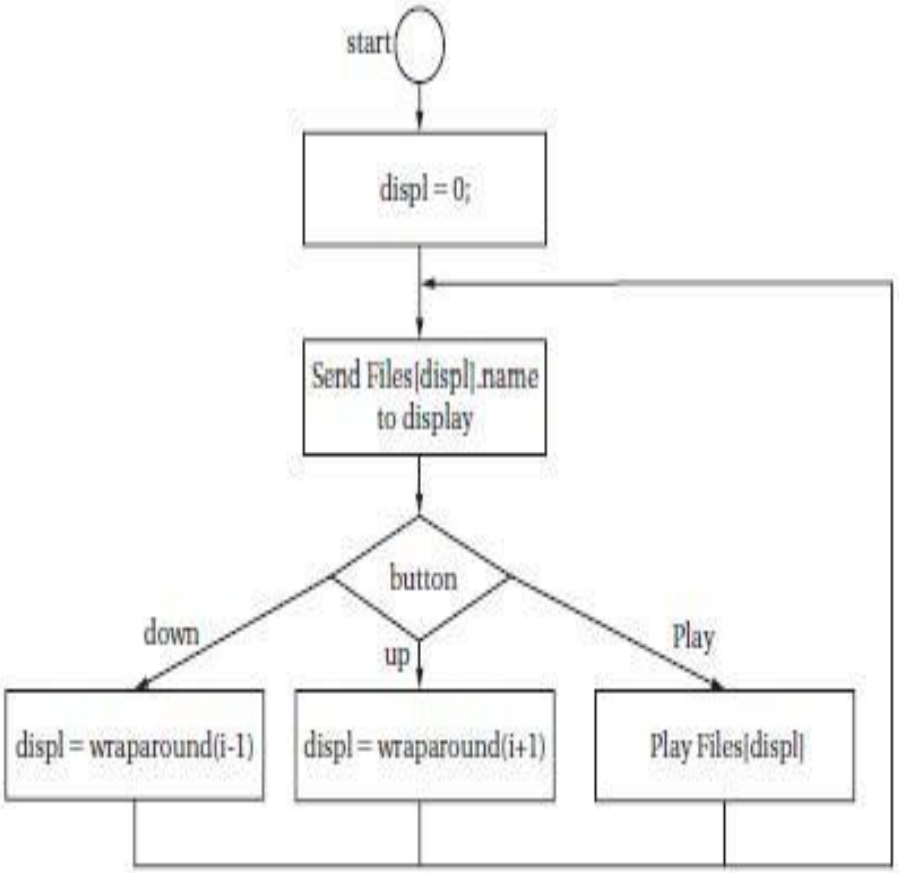
Specification

- The File ID class is an abstraction of a file in the flash file system.
- The controller class provides the method that operates the player.



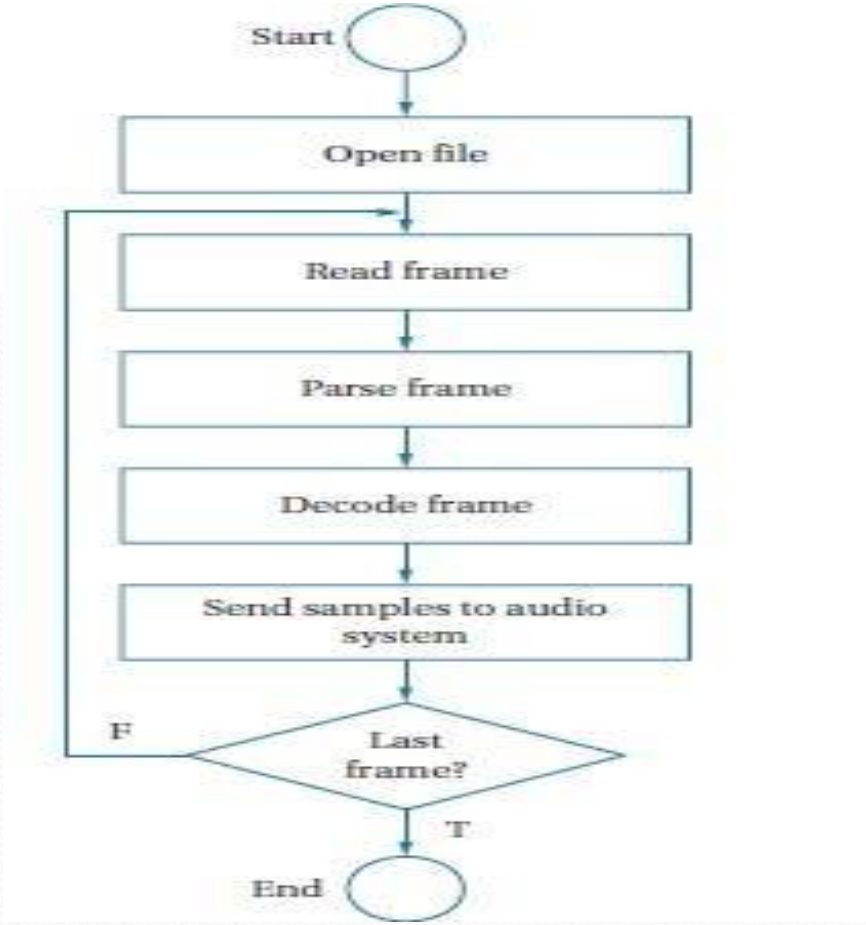
State diagram for file display and selection

- This specification assumes that all files are in the root directory and that all files are playable audio.



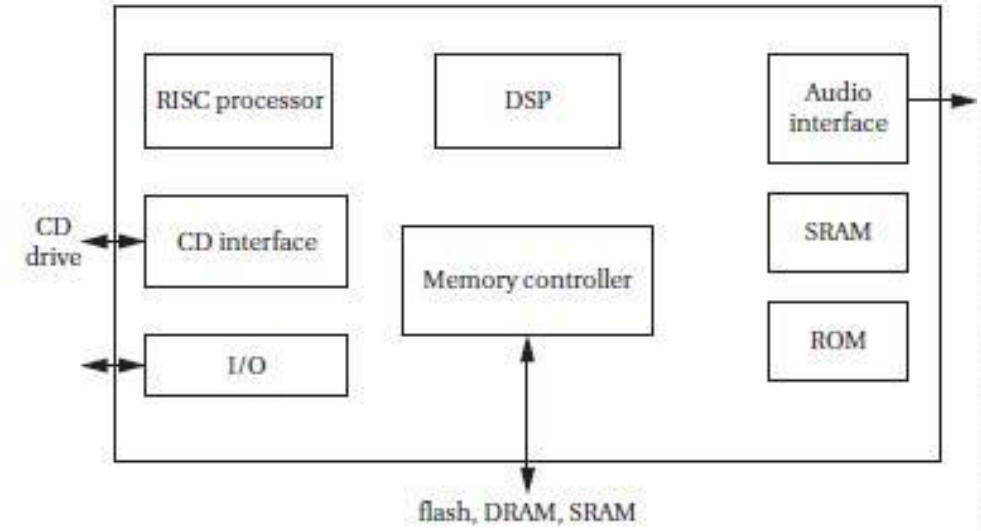
State diagram for Audio Playback

- It refers to sending the samples to the audio system.
- Playback and reading the next data frame must be overlapped to ensure continuous operation.
- The details of playback depend on the hardware platform selected, but will probably involve a DMA transfer.



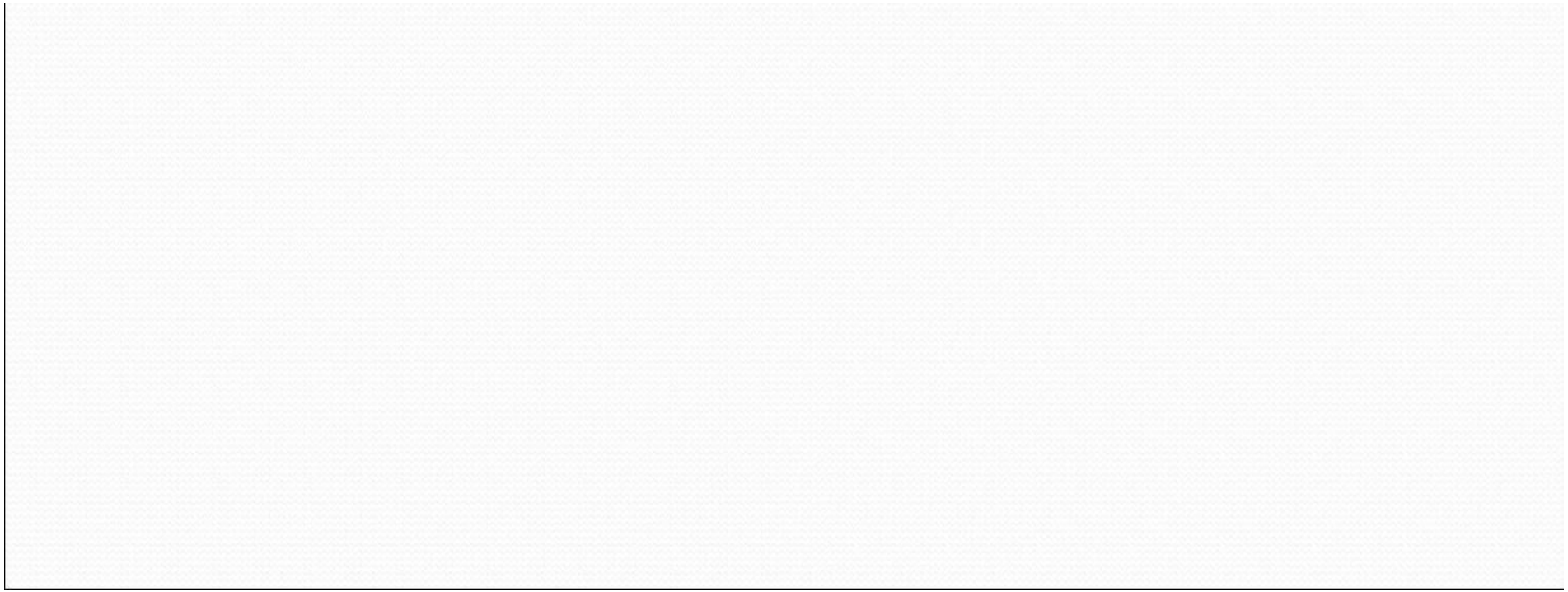
System architecture

- The audio controller includes two processors.
- The 32-bit RISC processor is used to perform system control and audio decoding.
- The 16-bit DSP is used to perform audio effects such as equalization.
- The memory controller can be interfaced to several different types of memory.
- Flash memory can be used for data or code storage.
- DRAM can be used to handle temporary disruptions of the CD data stream.
- The audio interface unit puts out audio in formats that can be used by A/D converters.
- General- purpose I/O pins can be used to decode buttons, run displays.



Component design and testing

- The audio output system should be tested separately from the compression system.
- Testing of audio decompression requires sample audio files.
- The standard file system can either implement in a DOS FAT or a new file system.
- While a non-standard file system may be easier to implement on the device, it also requires software to create the file system.
- The file system and user interface can be tested independently .



System integration and debugging

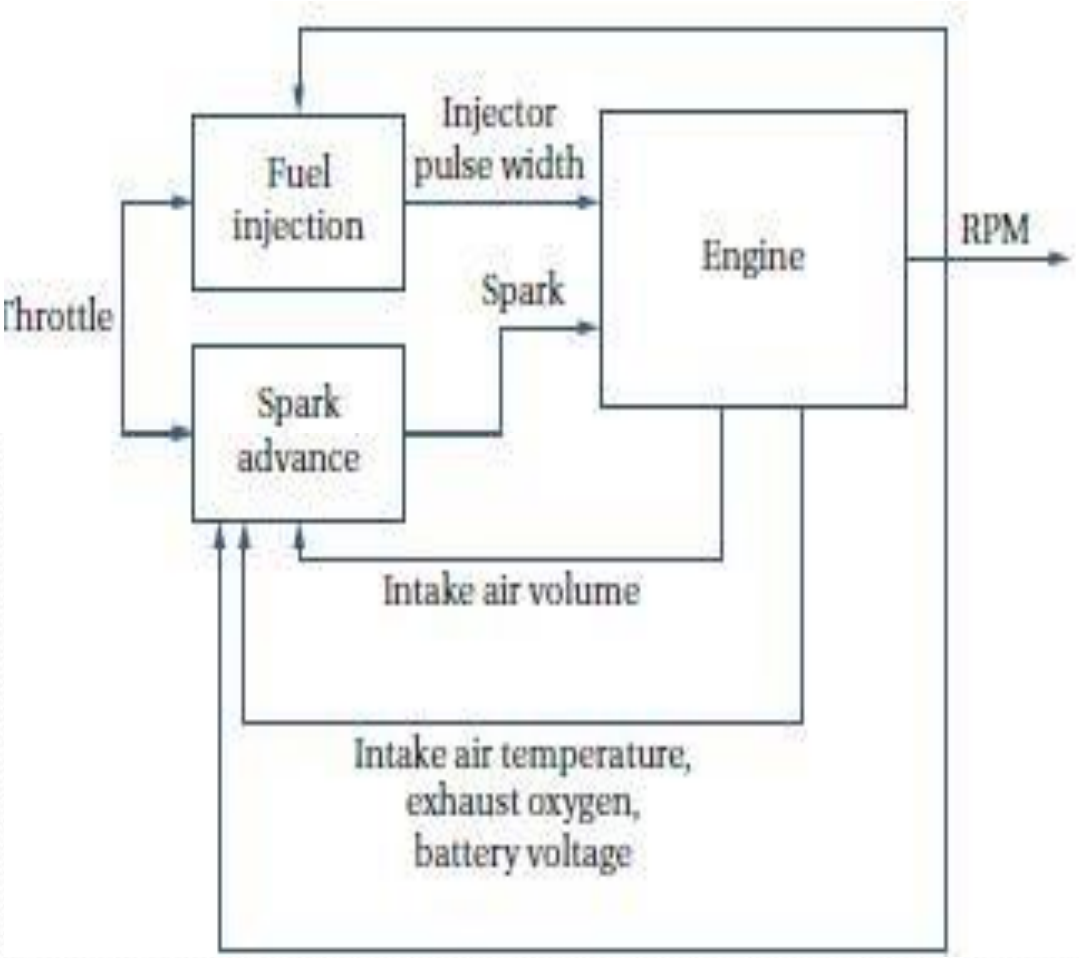
- It ensure that audio plays smoothly and without interruption.
- Any file access and audio output that operate concurrently should be separately tested, ideally using an easily recognizable test signal.

Engine Control Unit

- This unit controls the operation of a fuel-injected engine based on several measurements taken from the running engine.

Operation and Requirements

- The throttle is the command input.
- The engine measures throttle, RPM, intake air volume, and other variables.
- The engine controller computes injector pulse width and spark.



Requirements

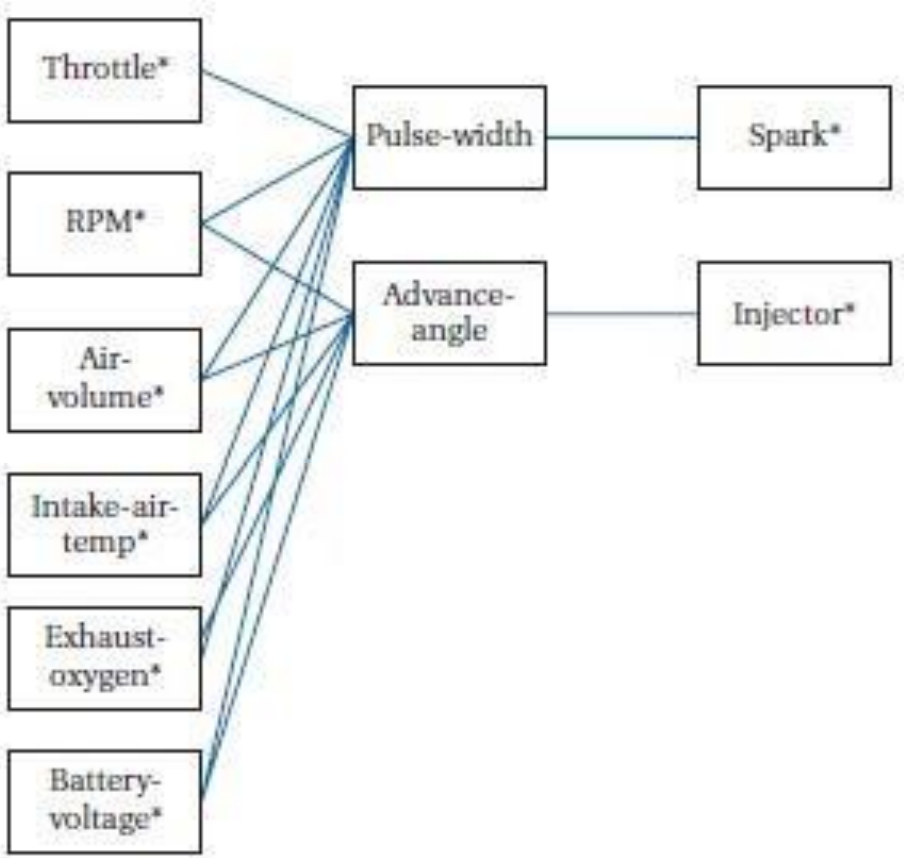
Name	ECU
Purpose	Engine controller for fuel-injected engine
Inputs	Throttle, RPM, intake air volume, intake manifold pressure
Outputs	Injector pulse width, spark advance angle
Functions	Compute injector pulse width and spark advance angle as a function of throttle, RPM, intake air volume, intake manifold pressure
Performance	Injector pulse updated at 2-ms period, spark advance angle updated at 1-ms period
Manufacturing cost	Approximately $50
Power	Powered by engine generator
Physical size and weight	Approx 4 in × 4 in, less than 1 pound.

Specification

- The engine controller must deal with processes at different rates
- ΔNE and ΔT to represent the change in RPM and throttle position.
- Controller computes two output signals, injector pulse width PW and spark advance angle S.
  - $S=k_2X\ \Delta NE-k_3VS$
- The controller then applies corrections to these initial values
- If intake air temperature (THA) increases during engine warm-up, the controller reduces the injection duration.
- If the throttle opens, the controller temporarily increases the injection frequency.
- Controller adjusts duration up or down based upon readings from the exhaust oxygen sensor (OX).

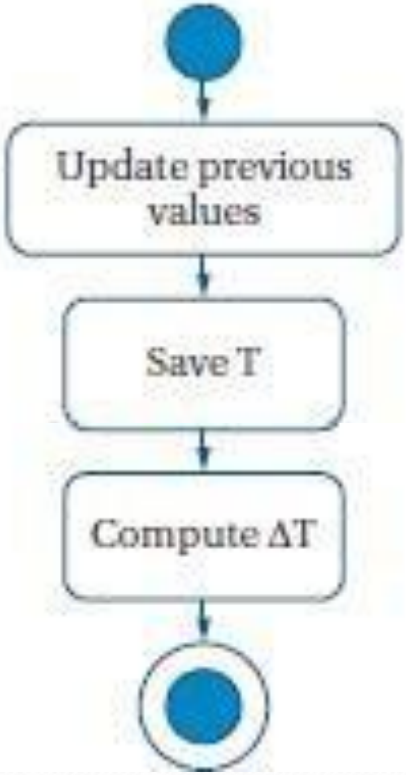
System architecture

- The two major processes, pulse-width and advance-angle, compute the control parameters for the spark plugs and injectors.
- Control parameters rely on changes in some of the input signals.
- Physical sensor classes used to compute these values.
- Each change must be updated at the variable's sampling rate.



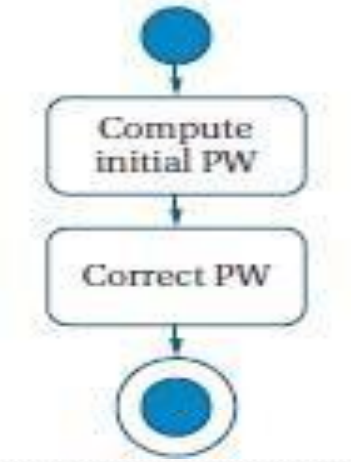
State diagram for throttle position sensing

- Throttle sensing, which saves both the current value and change in value of the throttle.

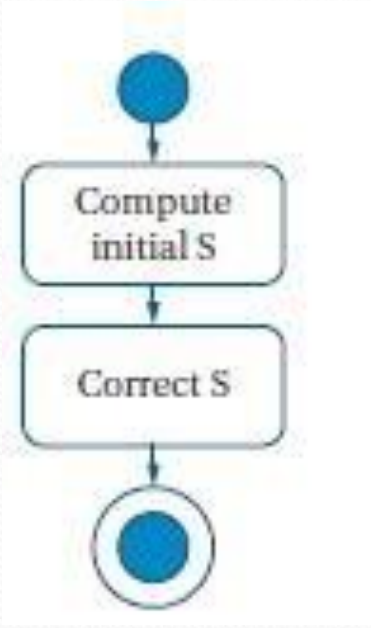


State diagram for injector pulsewidth

- In each case, the value is computed in two stages, first an initial value followed by a correction.

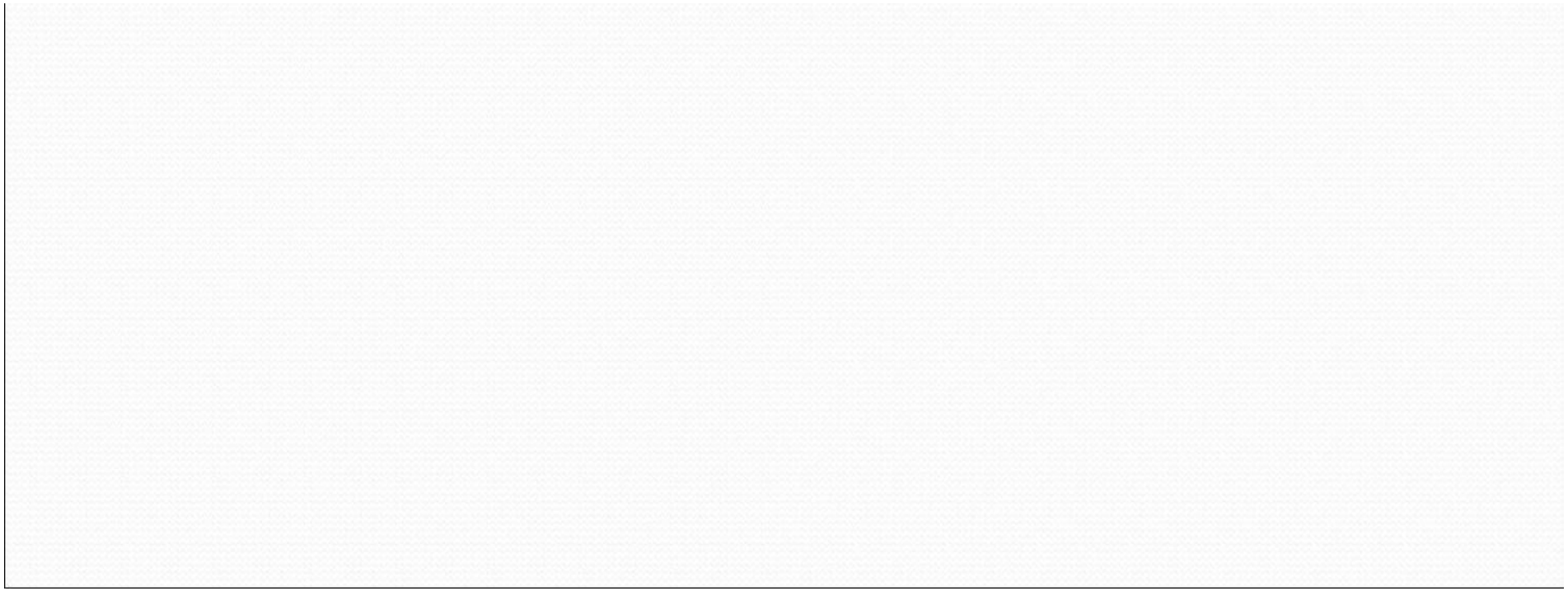


State diagram for spark advance angle



Component design and testing

- Various tasks must be coded to satisfy the requirements of RTOS processes.
- Variables that are maintained across task execution, such as the change-of-state variables, must be allocated and saved in appropriate memory locations.
- Some of the output variables depend on changes in state, these tasks should be tested with multiple input variable sequences to ensure that both the basic and adjustment calculations are performed correctly.

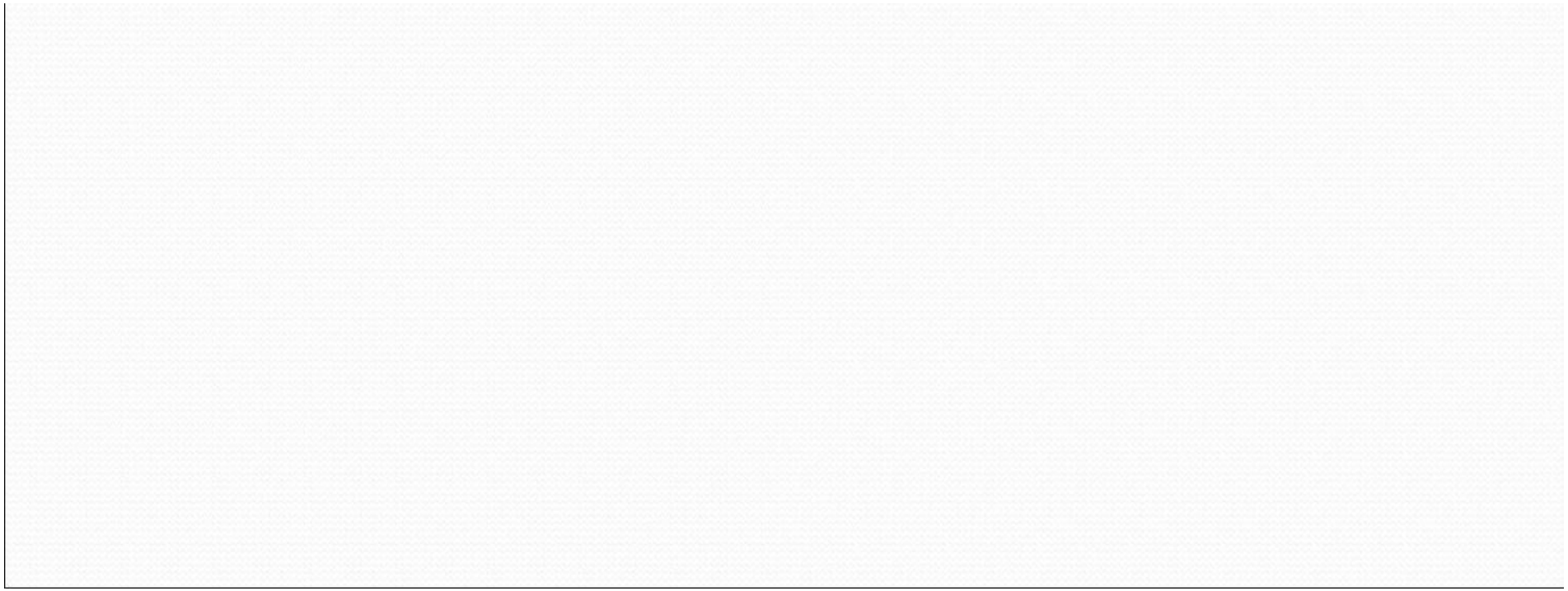


System integration and testing

- Engines generate huge amounts of electrical noise that can cripple digital electronics.
- They also operate over very wide temperature ranges.

1. hot during engine operation,
2. potentially very cold before the engine is started.

- Any testing performed on an actual engine must be conducted using an engine controller that has been designed to withstand the harsh environment of the engine compartment.



Video Accelerator

- It is a hardware circuits on a display adapter that speed up fill motion video.
- Primary video accelerator functions are color space conversion, which converts YUV to RGB.
- Hardware scaling is used to enlarge the image to full screen and double buffering which moves the frames into the frame buffer faster.

Video compression

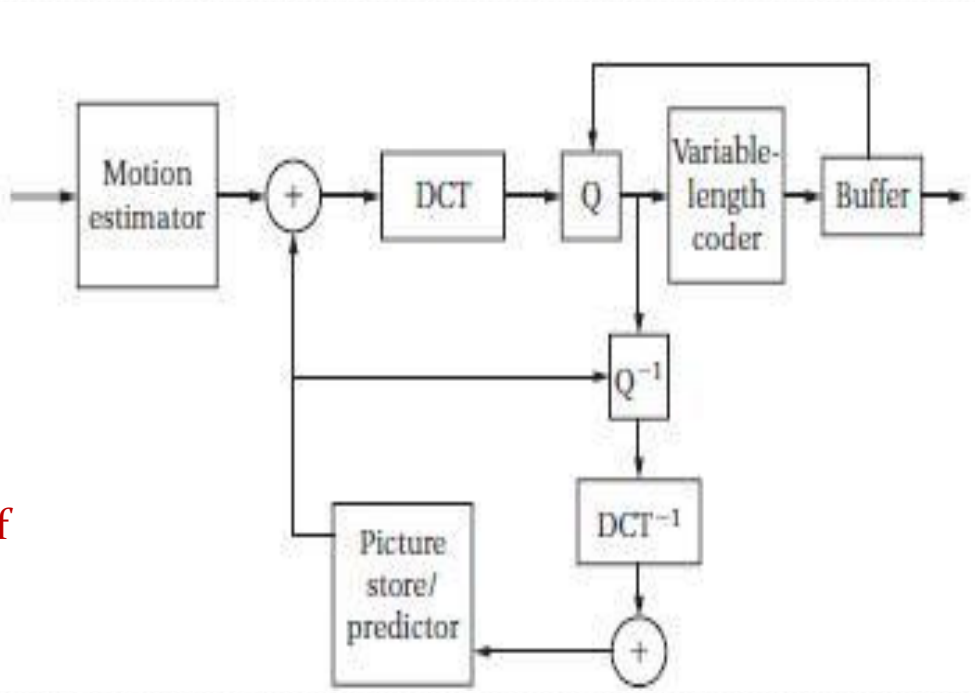
•MPEG-2 forms the basis for U.S. HDTV broadcasting.

• This compression uses several component algorithms together in a feedback loop.

•Discrete cosine transform (DCT) used in JPEG and MPEG-2.

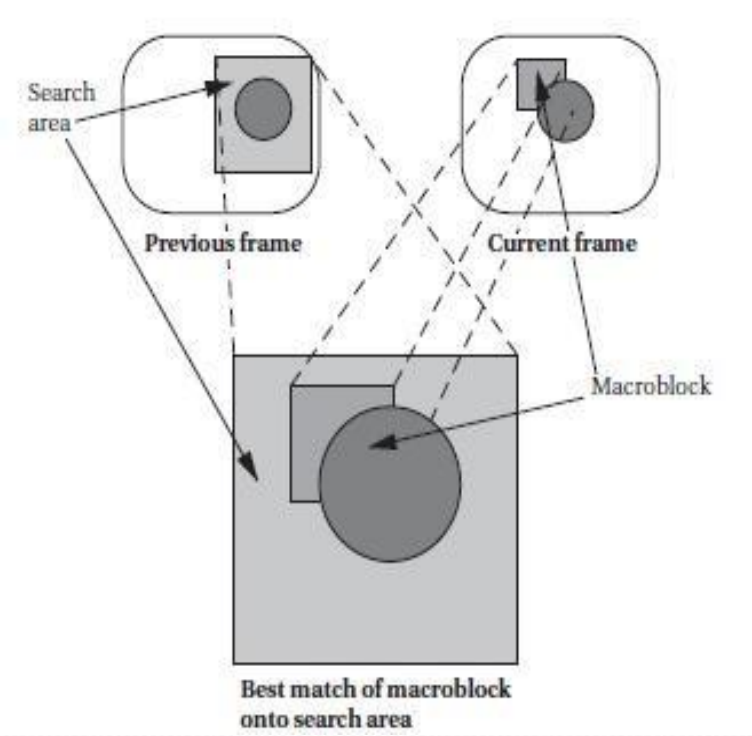
•DCT used a block of pixels which is quantized for lossy compression.

•Variable-length coder→assign number of bits required to represent the block.



Block motion Estimation

- MPEG uses motion to encode one frame in terms of another.
- Block motion estimation→some frames are sent as modified forms of other frames
- During encoding, the frame is divided into macro blocks.
- Encoder uses the encoding information to recreate the lossily-encoded picture, compares it to the original frame, and generates an error signal.
- Decoder keep recently decoded frames in memory so that it can retrieve the pixel values of macro-blocks.



5.13.2).Concept of Block motion estimation

- To find the best match between regions in the two frames.
- Divide the current frame into 16 x 16 macro blocks.
- For every macro block in the frame, to find the region in the previous frame that most closely matches the macro block.
- Measure similarity using the following sum-of-differences measure

$$\sum_{1 \le i,j \le n} |M(i,j) - S(i - o_x, j - o_y)|$$

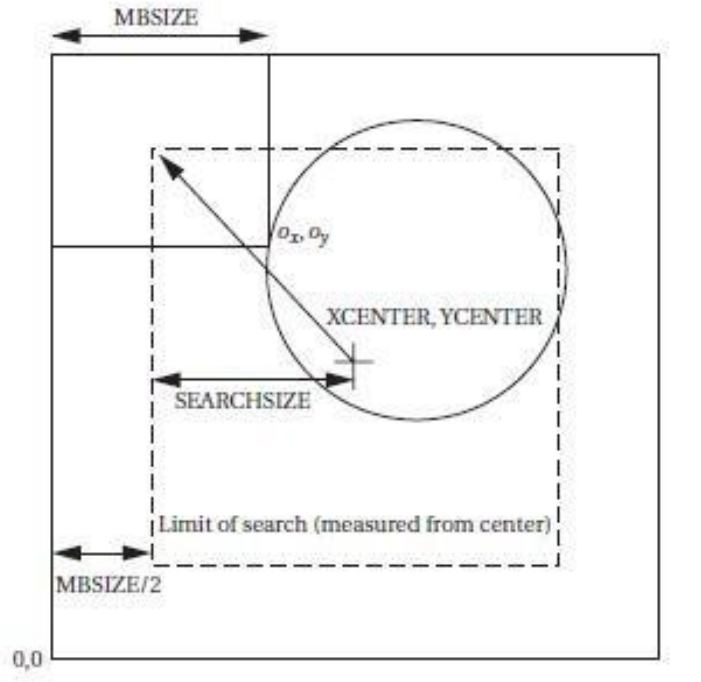
- M(i,j) → intensity of the macro block at pixel i,j,
- S(i,j) → intensity of the search region
- N→ size of the macro block in one dimension
- <ox, oy>→offset between the macro block and search region
- We choose the macro block position relative to the search area that gives us the smallest value for this metric.
- The offset at this chosen position describes a vector from the search area center to the macro block's center that is called the motion vector.

Algorithm and requirements

- C code for a single search, which assumes that the search region does not extend past the boundary of the frame.
- The arithmetic on each pixel is simple, but we have to process a lot of pixels.
- If MBSIZE is 16 and SEARCHSIZE is 8, and remembering that the search distance in each
- dimension is 8 + 1 + 8, then we must perform

$$n_{ops} = (16 \times 16) \times (17 \times 17) = 73{,}984$$

```
bestx = 0; besty = 0; /* initialize best location--none yet */
bestsad = MAXSAD; /* best sum-of--difference thus far */
for (ox = -SEARCHSIZE; ox < SEARCHSIZE; ox++) {
    /* x search ordinate */
    for (oy = -SEARCHSIZE; oy < SEARCHSIZE; oy++) {
        /* y search ordinate */
        int result = 0;
        for (i = 0; i < MBSIZE; i++) {
            for (j = 0; j < MBSIZE; j++) {
                result = result + iabs(mb[i][j] -
                    search[i -ox + XCENTER][j - oy + YCENTER]);
            }
        }
        if (result <= bestsad) { /* found better match */
            bestsad = result;
            bestx = ox; besty = oy;
        }
    }
}
```

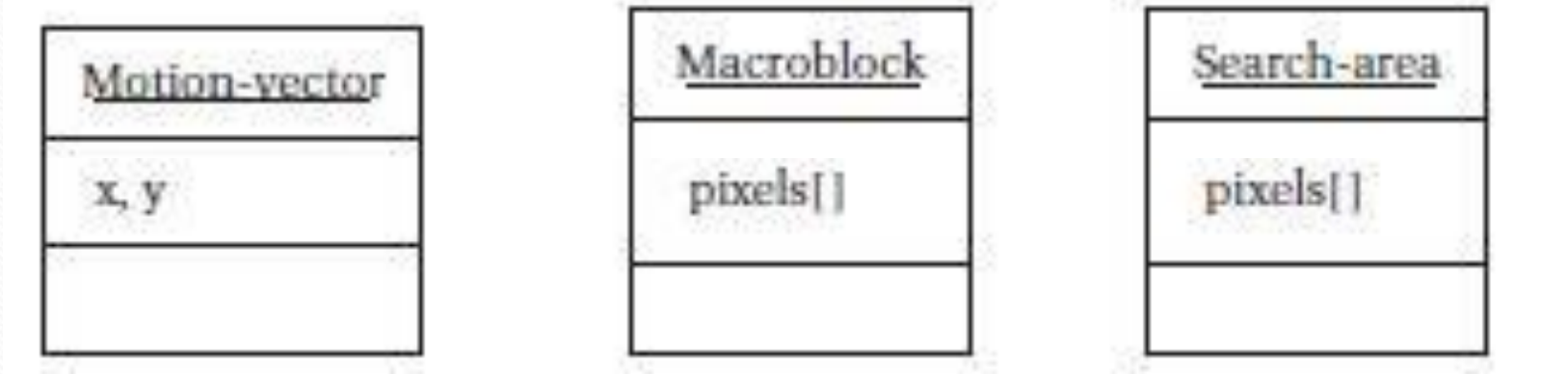


Requirements

Name	Block motion estimator
Purpose	Perform block motion estimation within a PC system
Inputs	Macroblocks and search areas
Outputs	Motion vectors
Functions	Compute motion vectors using full search algorithms
Performance	As fast as we can get
Manufacturing cost	Hundreds of dollars
Power	Powered by PC power supply
Physical size and weight	Packaged as PCI card for PC

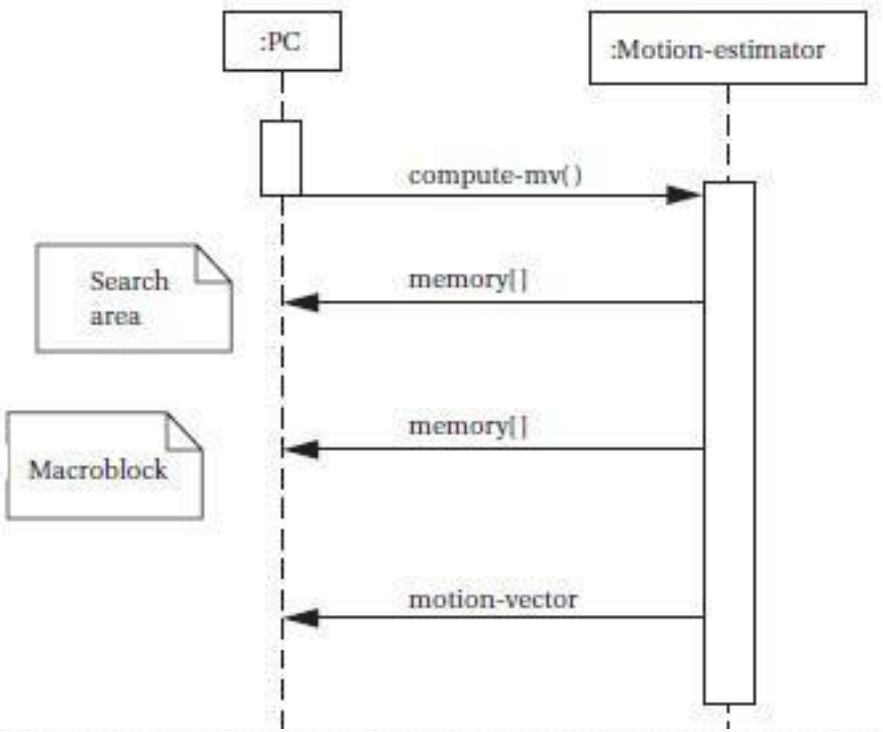
Specification

- Specification for the system is relatively straightforward because the algorithm is simple.
- The following classes used to describe basic data types in the system motion vector, macro block, search area.



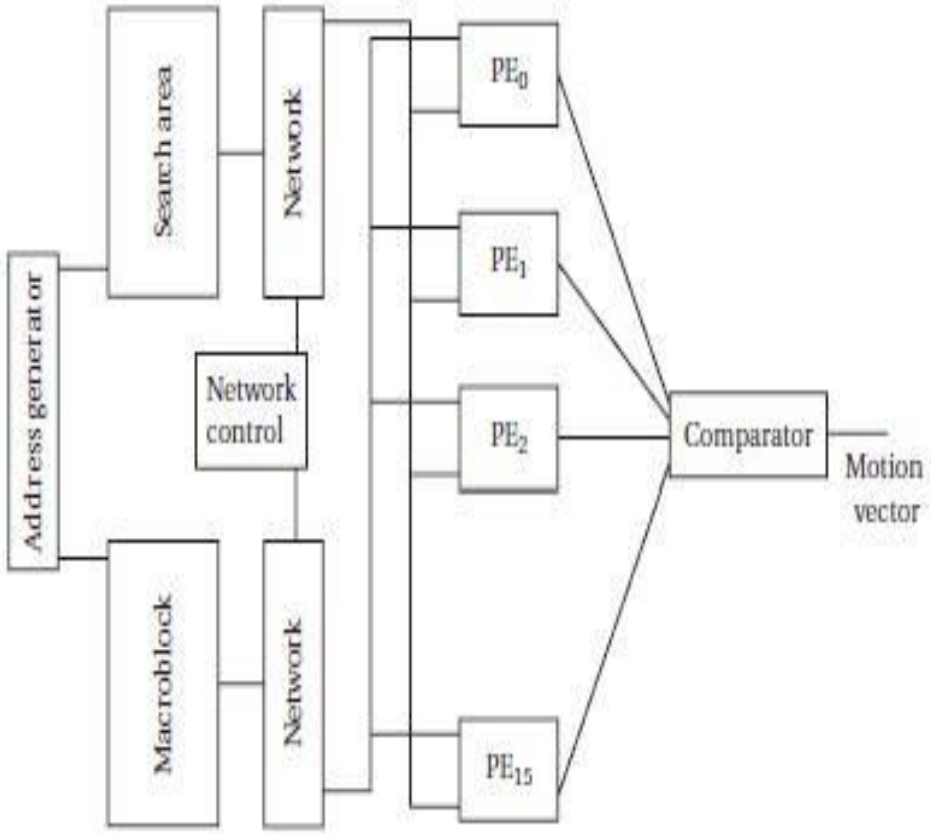
Sequence Diagram

- The accelerator provides a behavior compute-mv() that performs the block motion estimation algorithm.
- After initiating the behavior, the accelerator reads the search area and macro block from the PC, after computing the motion vector, it returns it to the PC.



Architecture

- The macro block has 16 x16 = 256.
- The search area has $(8 + 8 + 1 + 8 + 8)^2 =$ 1,089 pixels.
- FPGA probably will not have enough memory to hold 1,089 (8-bit)values.
- The machine has two memories, one for the macro block and another for the search memories.
- It has 16 processing elements that perform the difference calculation on a pair of pixels.
- Comparator sums them up and selects the best value to find the motion vector.



System testing

- Testing video algorithms requires a large amount of data.
- we are designing only a motion estimation accelerator and not a complete video compressor, it is probably easiest to use images, not video, for test data.
- use standard video tools to extract a few frames from a digitized video and store them in JPEG format.
- Open source for JPEG encoders and decoders is available.
- These programs can be modified to read JPEG images and put out pixels in the format required by your accelerator.