Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: A1

Title: Simulate the operations of LIFT.

Aim: Develop an application using Beeglebone Black/ ARM Cortex A5 development board to simulate the operations of LIFT.

Prerequisites:

- Programming in Python/C++
- Basics and Logical concerns of LIFT

Objectives:

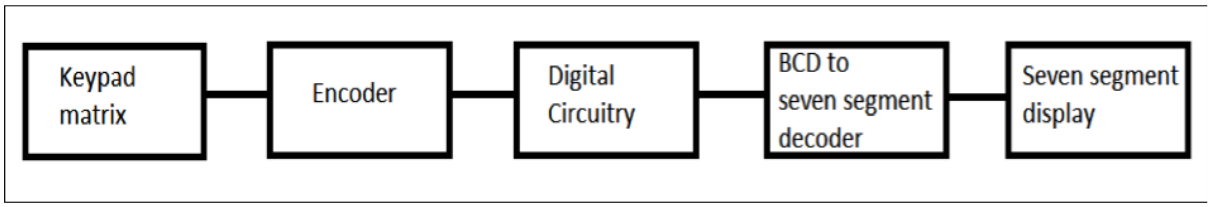
- To study embedded systems
- To study BeegleBone /ARM Cortex A5 development board

Theory:

Introduction:

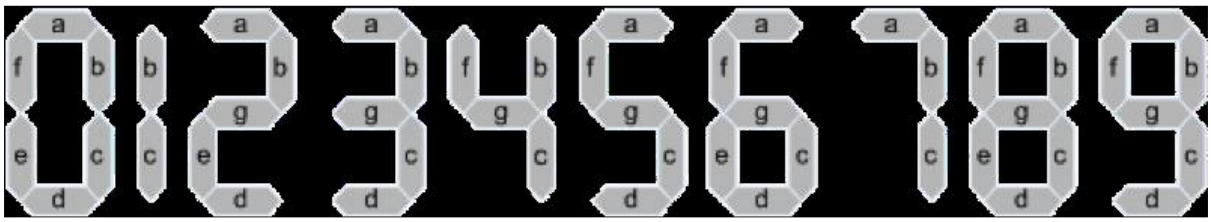
Ever wondered how small things in our lives go unnoticed at times. In today's world of skyscrapers, crossing a hundred floors in a matter of few minutes is no big deal, thanks to the Elevators! But no, we will not be talking about how an elevator works as it is primarily guided by principles that are better understood to the mechanical engineers. The whole set-up is rather complex with many control systems and processors forming the core of the elevator scheme. In this discussion, we will deal with the role of electronics in the modern-day elevator system the display control.

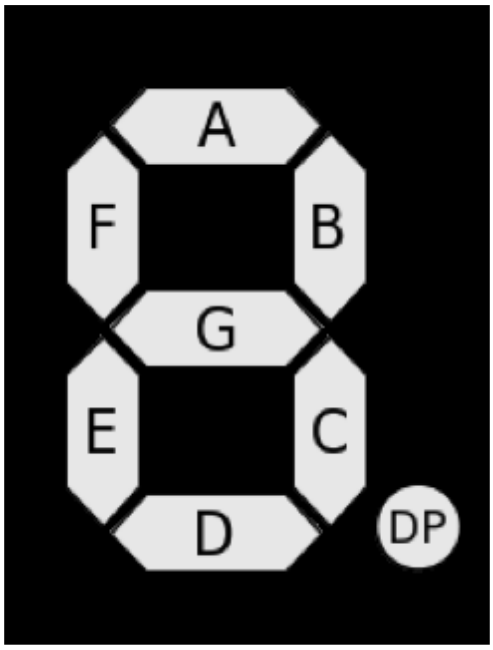
For simplicity, we will only be dealing with the fundamental circuitry or concept underlying the displays employed. The actual connections or circuits may-be larger and heavily complicated. To give you an idea, here is a block diagram you can identify with.



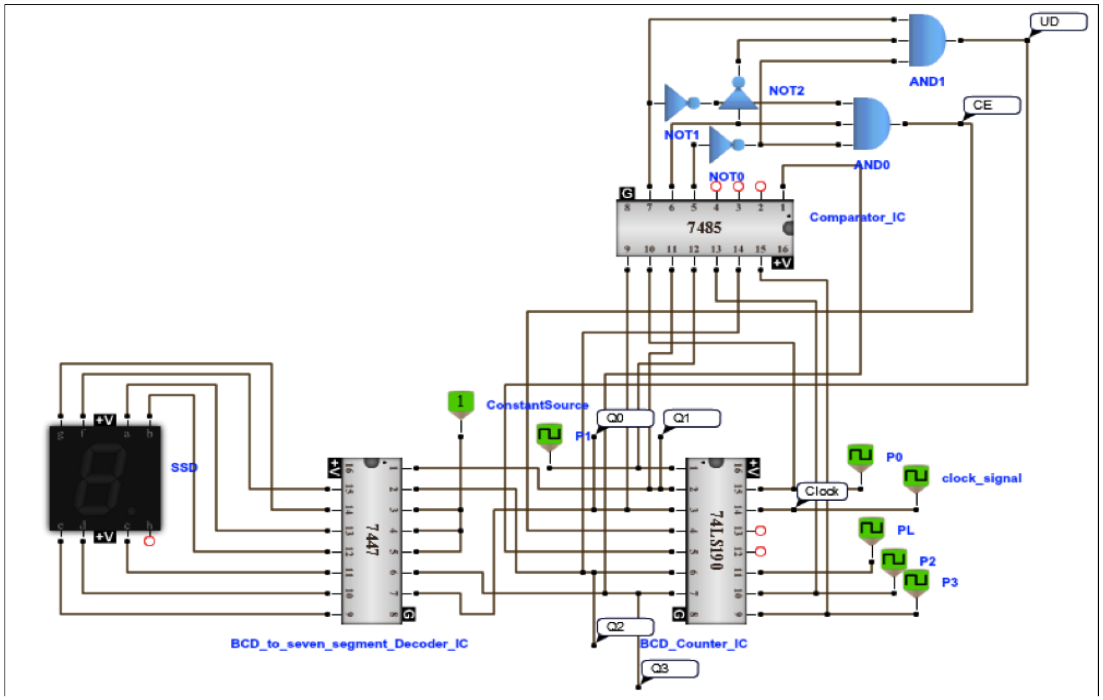
What exactly are we talking about? Here we will dig into the details behind the displays that are synchronized with the lift movement, more precisely, how does the display show "1" when we are currently at the first floor and so forth.

Coming back to the block diagram, let us first define what a keypad matrix is. You can consider it to be the same as the buttons you see in the elevator: 0-1-2-3-4-5-6-7-8-9 and so on. When you press on a particular number, say "1", a signal is sent to the processing unit that identifies the input as "1" and accordingly generates the corresponding output. So this is how we take the input from the user. The next part is to process this unit in a form that is better understood by the processor. For this we employ an encoder that converts "1" into "0001" (binary) and sends it to the next block for further processing. After the processing has been done, it is time to display the data to the user for which a device called the seven segment display or SSD is used. Now, it is the same device that displays time in a digital clock but sadly it cannot be directly connected to the core circuitry. The SSD contains a set of seven LEDs, namely a-b-c-d-e-f-g-h that light up in accordance to the input to show 0-1-2-3-4-5-6-7-8-9. Have a look:



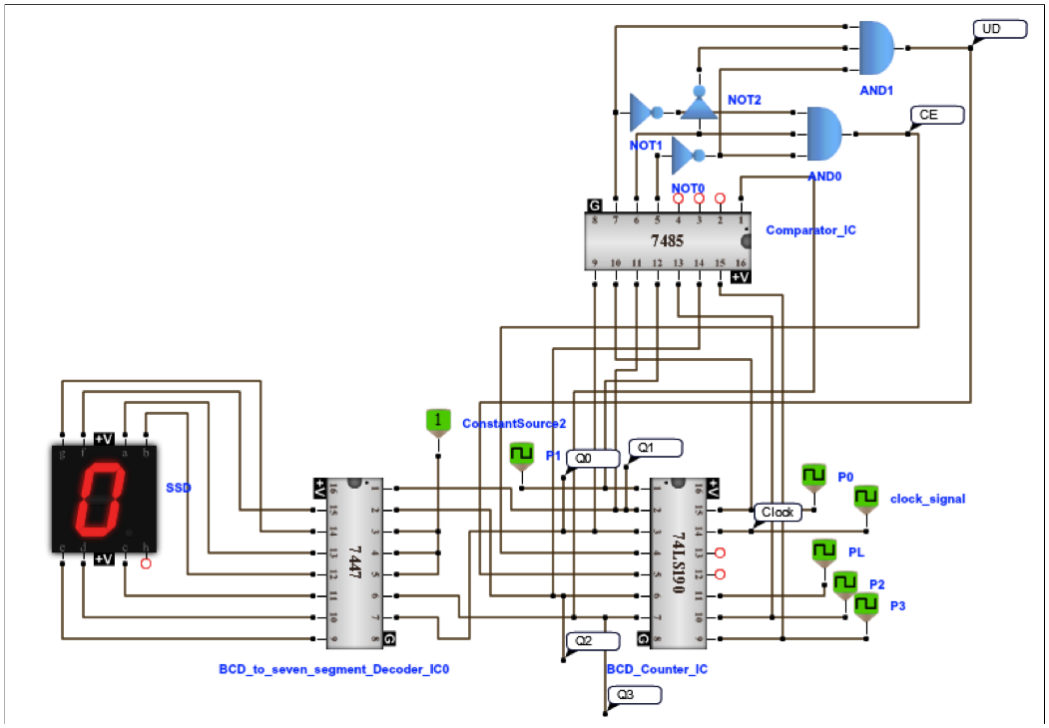


For this the connections are made via a BCD-to-seven segment decoder that decodes the input into a format that is understood by the SSD. The format of the data output coming from the counter will be in the form of binary coded decimal or BCD. IC 7447 is most commonly used for this decoding purpose. It accepts BCD and subsequently assigns a logic 1 to the LEDs that should be glowing in order to display that particular BCD, for example, if the input BCD is "0001" the decoder output will be a=1,b=0,c=1,d=1,e=1,f=1,g=1. This enables the SSD to deduce that the number 6 is to be displayed. This is how all the digits from 0 to 9 can be displayed using a single SSD. Coming back to the processing block, it can be explained with the help of a situational example: Suppose you enter a multi-storey building and wish to go to the 3rd floor. Your first action will be to press the button at the ground floor and wait for the lift to come down if the lift is not at the ground floor initially. Once you enter the lift, you will press the button "3" and read the display that shows: 0-1-2-3, and voila! After the lift has dropped you at the 3rd floor, it will once again go back to the ground floor i.e. now the display will be something like this: 3-2-1-0. So, how does this counting take place? It's simple, it uses a counter. Are you asking yourself how could it be that easy? Have a look at the circuit designed as:

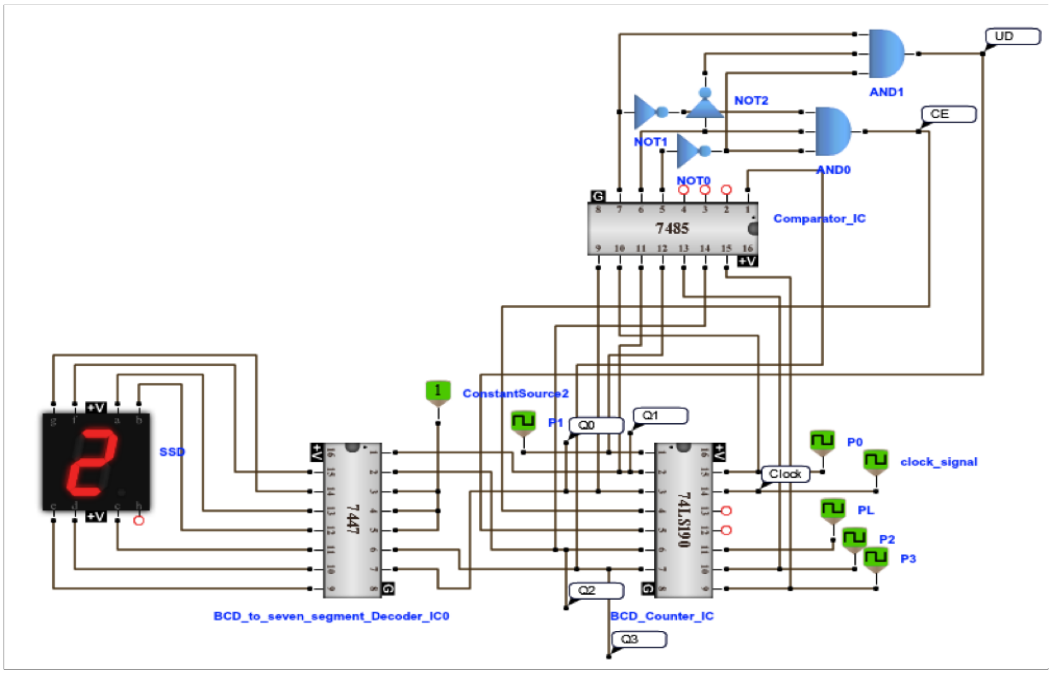


Elevator Control Circuit The circuit uses three ICs namely: 7485 which is a 4-bit magnitude comparator, 74190 which is an up-down decade counter and 7447 which is a BCD to seven-segment decoder IC. The interconnections made within the circuit would illustrate the working of the display system. First, the input given by the user (after binary encoding) is given as input to the decade counter from the parallel input pins P0, P1, P2, P3. The parallel load pin PL is active low and therefore is given logic 0. The output from the decade counter is passed to the magnitude comparator which is compared with the input from the P0, P1, P2, P3 pins. The comparator has three outputs viz: greater than, lesser than, equal to. The comparison may be any of these three possibilities and therefore it is essential to decide whether to keep counting up or start counting down based on the comparator output. This decision circuitry is designed as a combinational circuit with the help of logic gates. As can be seen in the circuit, the two outputs from the logic circuit are CE and UD. These are respectively connected to the CE and U/D pin of the counter IC. The CE pin is active low and is called count enable, therefore the counter counts every time this pin receives a logic '0'. The U/D pin dictates whether the counter is in up-counter (logic '0' at U/D pin) or down-counter (logic '1' at U/D pin) mode respectively. Finally, the output at each stage from the counter is given to the decoder IC that keeps displaying the corresponding floor number in the seven segment display. Carrying on with our example, the user input here is '3' (for third floor) or '0001' (binary output from the encoder). When this '0011' is fed into this counter IC as shown, the ordinary decade counter will not count up to 9 anymore, instead it will count up to 3. The initial floor is '0' and it moves up to the 3rd floor. Then we have simulated a user input of '1' -0001- at the parallel load input. This results in the display now going to '1' akin to the movement of the elevator to a lower floor from the upper floor.

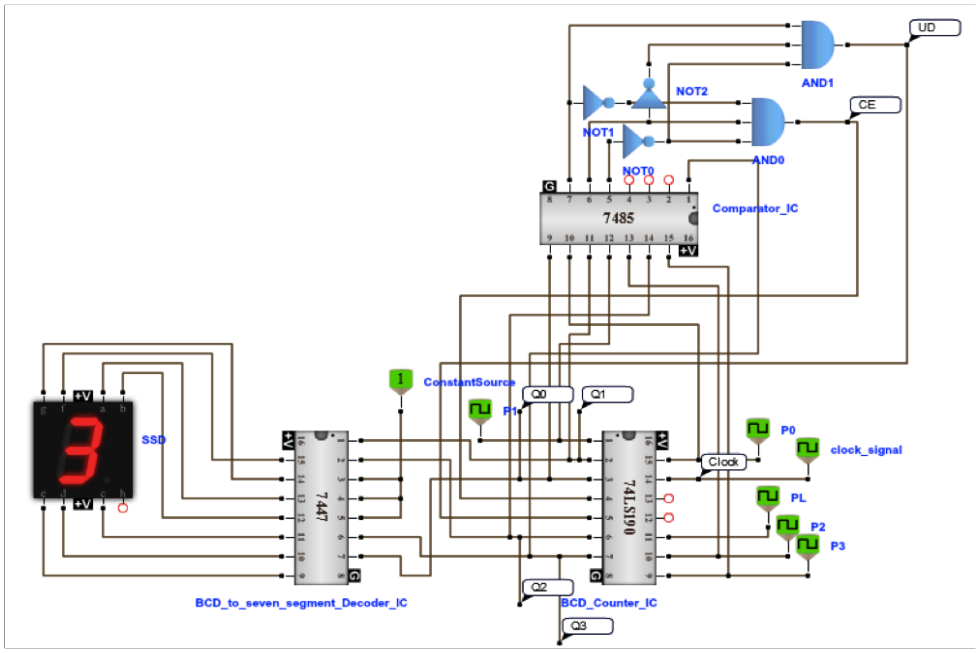
Initially:



During UP count:



During DOWN count:



Therefore the lift comes back to its ground state after every cycle unless a trigger is applied to call the elevator to some other floor. Not as easily done as said, but certainly the fundamentals remain the same. It is an irrefutable fact that some simple electronics runs at the back-end of almost everything that comprises today's modern world, be it the metro that displays station name and other vital information, your trendy digital watches, the cool display with stock market updates on Wall Street, or of course the elevators!

Conclusion:

We have successfully developed an application using Beeglebone Black/ ARM Cortex A5 development board to simulate the operations of LIFT.

FAQs:

1. Difference between RISC and CISC processor.
2. List various advanced features of ARM processor.
3. List and explain various operating modes of ARM processor.
4. Explain in short Register set of ARM processor.
5. Define Interrupt vector table.
6. Give list of interrupt vector table.
7. Explain in short Instruction set of ARM processor
8. What is Conditional execution in ARM processor
9. Define and Explain ARM assembly language program.
10. Give list of arithmetic instruction of ARM.
11. What is the use of TST instruction?
12. What is the use of TEQ instruction?
13. Write the general form of source lines in assembly language.
14. How is the literal pool accessed?
15. Write the syntax for load and store instructions.
16. Give list load and store instructions.
17. What is the maximum size of constant that can be used in immediate mode.
18. Explain read only and read write memory.
19. Write syntax LDM and STM instruction.
20. Explain ARM 9

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: A2

Title: Simulate the Working of Signal Lights.

Aim: Develop an application using Beeglebone Black/ ARM Cortex A5 development board to simulate the working of signal lights.

Prerequisites:

- Programming in Python/C++
- Basics and Logical concerns of Traffic Signal Lights

Objectives:

- To earn Basic concepts of BeegleBone and using an ARM Cortex A5 processor.
- Learning to work with a BeegleBone Black kit.
- To program a traffic light simulation on the kit.
- To observe the hardware interfacing of the BeegleBone kit with using suitable program and interfaces.

Theory:

Introduction of BeegleBone Black:

BeegleBone Black is the latest generation of low-power, single-board, highly-capable open-source hardware based on the ultra-efficient ARM processing platform. Produced by Circuitco and backed by Texas Instruments, BeegleBone Black was designed to be a community-supported, maker-centric technology. However, this small size, low cost form factor has shown the potential to cross over into the commercial space as companies like Element 14's Embest begin ramping up production of their own, BeagleBone Certified, version of the board. Rev C of the BeagleBone Black, with double the flash memory and Debian Linux onboard, will launch in May. Logic Supply engineers are hard at work exploring the potential of this exciting innovation by focusing on software development as well as enclosure, cape and board design for BeagleBone Black and other emerging ARM-based form factors.

The working model of a traffic light setup has been provided included and interfaces with a BeagleBone Black ARM processor. The whole set-up is rather complex with many control systems and processors forming the core of the elevator scheme. In this discussion, we will deal with the role of electronics in the modern-day elevator system - the display control. For simplicity, we will only be dealing with the fundamental circuitry or concept underlying the displays employed. The actual connections or circuits may-be larger and heavily complicated. To give you an idea, here is a block diagram you can identify with.

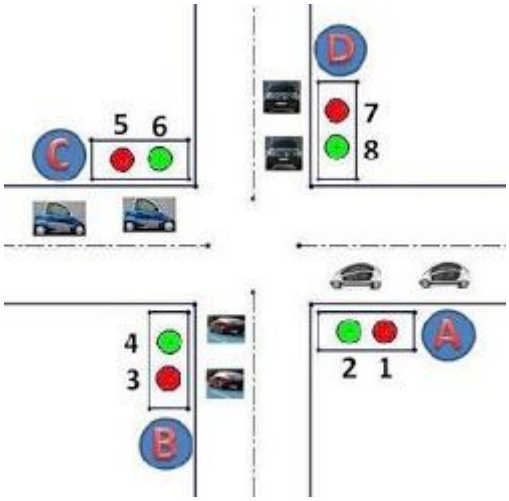
Traffic Light Controller Interface Kit:

- 4 junction Road, 4 set of Red, Green and Yellow LEDs
- Pedestrian Crossings bi-colour LEDs
- Control switch to control day and night operations
- Closed wooden cabinet

Traffic Light Controller Interface Kit:

Traffic Light Control:

Traffic lights, which may also be known as stoplights, traffic lamps, traffic signals, signal lights, robots or semaphore, are signaling devices positioned at road intersections, pedestrian crossings and other locations to control competing flows of traffic.



About the colors of Traffic Light Control:
Traffic lights alternate the right of way of road users by displaying lights of a standard color (red, yellow/amber, and green), using a universal color code (and a precise sequence to enable comprehension by those who are color blind).

In the typical sequence of colored lights:

- Illumination of the green light allows traffic to proceed in the direction denoted,
- Illumination of the yellow/amber light denoting, if safe to do so, prepare to stop short of the intersection, and
- Illumination of the red signal prohibits any traffic from proceeding.

Usually, the red light contains some orange in its hue, and the green light contains some blue, for the benefit of people with red-green color blindness, and "green" lights in many areas are in fact blue lenses on a yellow light (which together appear green).

Conclusion:
Thus, we have performed the required traffic lights simulation on a BeagleBone Black kit using the proper interfacing and sequencing of the lights.

FAQs:

1. What is traffic simulation?
2. What power peripheral is required for Beegalblack bone?
3. What is role of ARM cortex A5 processor?
4. How ARM cortex A5 processor sense the traffic
5. Give Specification for ARM cortex A5 processor
6. How do we interface beegalbone black with LED's
7. Give specification of Beegal black bone board
8. List out all features of ARM Cortex A5
9. List out all applications of beegal Black bone borad
10. State and explain purpose of Traffic signal system.
11. What are chaalenges to develop signal timing policies
12. What data should be collected to develop a signal timing
13. What should be measured after implementation
14. What should be monitored as a part of maintanince while developing traffic signal simulator
15. What performance measure should be tested?
16. What measures will be used to determine the timing plan is effective.
17. How to frequently will be signal timing plan be reviewd and updated
18. Draw Pin configuration for ARM cortex processor
19. List out Hardware requirement for simulating traffic signal system.
20. Demonstate accuracy of model simulated against existing situation

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: A3

Title: Implement calculator (64 bit Binary Multiplication) application using concurrent Lisp

Aim: To design a system for performing 64 bit mathematical operation in LISP, The concurrency is typically implemented in LISP using Thread programming.

Prerequisites:

Basics of Common Lisp, Concurrent Lisp

Objectives:

Student should be able to implement concept concurrent programming and use the concepts of concurrent LISP

Theory:

LISP:

The name LISP derives from "List Processing". Linked lists are one of Lisp language's major data structures, and Lisp source code is itself made up of lists. As a result, Lisp programs can manipulate source code as a data structure, giving rise to the macro systems that allow programmers to create new syntax or new domain-specific languages embedded in Lisp.

SBCL:

Steel Bank Common Lisp (SBCL) is a high performance Common Lisp compiler. It is open source / free software, with a permissive license. In addition to the compiler and runtime system for ANSI Common Lisp, it provides an interactive environment including a debugger, a statistical profiler, a code coverage tool, and many other extensions. SBCL runs on a number of POSIX platforms

Steel Bank Common Lisp meets the criteria:

- Provides an identical API for threading and concurrency control on all of these platforms.
- Runs on MacOS, Windows, and Linux (among other platforms)
- Has a comprehensive foreign function interface to allow interfacing with C
- It's a native Lisp -- no JVM, no bytecode
- It's well supported by SLIME and Quicklisp
- It's a mature and complete implementation

Dealing with threads in SBCL:

Threads are a wellknown technique for concurrent programming: Doing more than one thing at a time. SBCL supports threading by following ways.

— Structure: sb-thread:thread

Class precedence list: thread, structure-object, t

Thread type. Do not rely on threads being structs as it may change in future versions.

— Variable: sb-thread:*current-thread*

Bound in each thread to the thread itself.

— Function: sb-thread:make-thread function &key name

Create a new thread of name that runs function. When the function returns the thread exits.

— Function: sb-thread:thread-alive-p thread

Check if thread is running.

— Function: sb-thread:list-all-threads

Return a list of the live threads.

Facilities:

SBCL support, Latest version of 64 Bit Linux Operating Systems (Ubuntu)

Algorithm:

1. Start
2. Read two numbers
3. Perform arithmetic operation
4. Display output
5. Stop

Input:

Two integer numbers

Output:

Integer Value

Conclusion:

We learned concept concurrent programming using concurrent LISP.

Questions:

1. What is Concurrent LISP?
2. Explain Concurrent Programming.
3. Differentiate Concurrent, Distributed and Parallel Programming.
4. What are the basic building blocks of LISP?
5. What is SBCL?
6. What is parallel evaluation mechanisms of concurrent LISP
7. What are the applications of LISP?

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: A5

Title: Project plan, SRS, Design document and Test Plan for Robotics (stepper motor) Application using Beagle Board.

Aim: Create Project plan, SRS, Design document and Test Plan for one group-C assignment from embedded operating system or Concurrent and Distributed Programming

Prerequisites:

- Concept of Software Engineering
- Concept of basics of Project plan and Design document

Objectives:

- Ability to Design document required for developing software project.
- Ability to do SRS and Test Plan.

Theory:

Project Plan:
Project planning is a discipline for stating how to complete a project within a certain timeframe. A project plan, according to the Project Management Body of Knowledge, is: "A formal, approved document used to guide both project execution and project control". The primary uses of the project plan are to document planning assumptions and decisions, facilitate communication among stakeholders, and document approved scope, cost, and schedule baselines. A project plan may be summarized or detailed. The objective of a project plan is to define the approach to be used by the Project team to deliver the intended project management scope of the project. At a minimum, a project plan answers basic questions about the project:

- **Why?** - What is the problem or value proposition addressed by the project? Why is it being sponsored?
- **What?** - What is the work that will be performed on the project? What are the major products/deliverables?
- **Who?** - Who will be involved and what will be their responsibilities within the project? How will they be organized?
- **When?** - What is the project timeline and when will particularly meaningful points, referred to as milestones, be complete?

- **Project plan for Stepper Motor is as follows:**

Category	**Recommended Action Item**	**Time span**
1.Planning and gathering requirement	• Make algorithm for project • Requirement include Beagle Bone and System with required software	1 Hr
2. System	• Start to create the project. • Generate the code.	4 hrs
3. Beagle Bone Black	• Create a connection between system & beagle board. • Check whether the beagle board successfully installed or not?	2 hrs
4. Stepper Motor	• Create a connection between beagle board and stepper motor.	15 min.
5. Testing of Project	• Testing of connection with Beagle bone and Stepper motor • Testing Results	20 Min

SRS (Software Requirements Specifications):-

Software Requirements Specification (SRS) is a perfect detailed description of the behavior of the system to be developed. That is SRS document is an agreement between the developer and the customer covering the functional and non-functional requirements of the software to be developed. SRS is considered as a contract between the customer and the developer. This SRS document will be used for verifying whether all the functional and non-functional requirements specified in the SRS are implemented in the product. The complete description of the functions to be performed by the software specified in the SRS will assist the potential users to determine if the software specified meets their needs or how the software must be modified to meet their needs.

Flexibility:
Flexibility is the first software requirement specification for any project. The flexibility of the stepper motor depends on frequency of the stepper motor. If the frequency is increased rotation of the stepper motor is increased. And the frequency is decreased then the rotation of the stepper motor is also decreased.

Performance:
Performance of the system is depends on the working of all the hardware and software components of the system which are used in project developments. As well as performance of the system also depends on code, this is used for developing the project. The code should be more flexible, scalable, and easy to implement.

Design constraints imposed on an implementation:

a) Hardware Requirements:

1. PC(x86 Architecture)
2. Beagle Bone Black
3. Stepper Motor Board
4. Power supply cable 5v and 12v

b) Software Requirements:

1. Beagle Bone Drives
2. Power Supply
3. Operating System
4. gcc compiler on target for direct compilation of source.

General Technical Specifications:
1) Hardware:

Supply, installation, commissioning and maintenance of all necessary hardware and networking equipment's and its connectivity as specified in the detailed specification. As a part of the project, the vendor should procure the required hardware and build the infrastructure as detailed in the Specifications. The vendor shall take the responsibility to install the servers, switches, routers, backup and tape devices, Workstation PCs, and other necessary hardware/software at the sites defined in the bid proposal sheet.

2) Networking:

The scope of work covers supply, installation, commissioning and maintenance of LAN at datacenter, Customer care centers, Sub division, division, Circle, Head Quarter and any other office of the utility as per their requirement along with creation of VPN/ MPLS based WAN solution. The vendor shall also provide the necessary drawings and plan for installation, sizing, cabling and connectivity and the bill of material for the networking of all the locations specified herein.

3) GIS System Software and maps:

Supply, installation, commissioning and maintenance of GIS software and latest

satellites imagery maps for GIS based customer indexing and asset mapping in the specified town. The successful bidder shall provide the maps along with the certificate and rights in favor of owner from the source that these are the latest as on date of purchase which should be later to award date.

Design Document:-

The following documents (one set each) will be required for smooth functioning of the system at data center and DR center: The successful vendor will provide ongoing product information for referential purposes and facilitating self-education by Utility personnel. Key aspects that the vendor will be evaluated on but not limited to include: What documentation is included in the standard license fee, for example:

- User manuals;
- Technical manuals;
- Installation guides;
- Business process guides;
- Program flow descriptions;
- Data model descriptions;
- Sample reports;
- Screen formats;
- Toolkit guides;
- Troubleshooting guides;
- Frequently asked question (FAQ) guides.

Introduction of Stepper Motor Basics:

Stepper motors consist of a permanent magnetic rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator. Following illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.

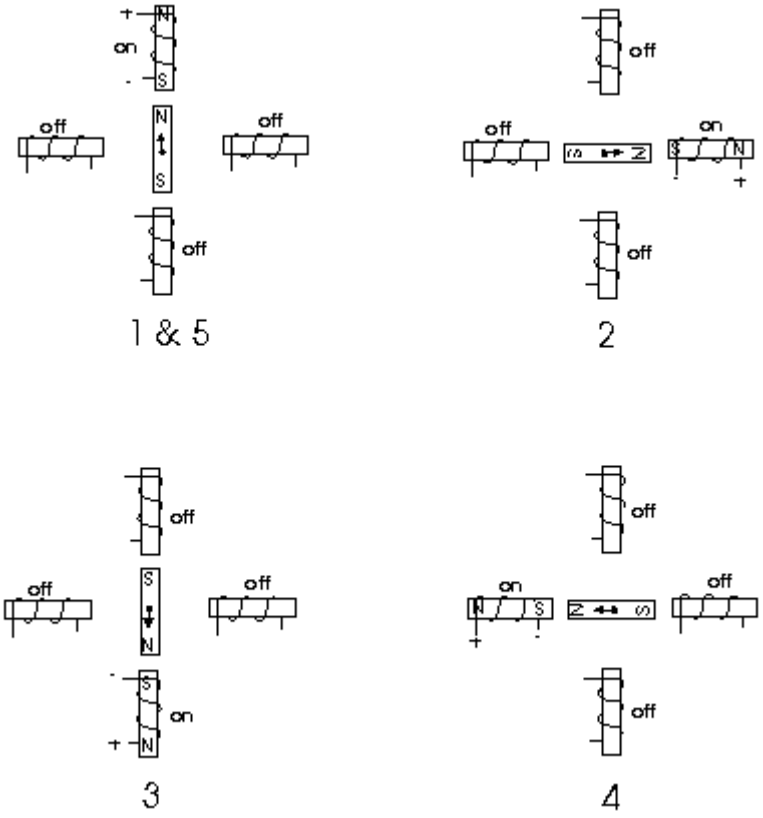


Table: Working off step motor

D	C	B	A
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0

1: Stator is magnetized
0: Stator in the normal state

This configuration is shown in table

Connector slot no.	BBB PIN no.	PIN Description	PIN connects to stepper motor connected FRC	Function
P9	1,2	GND	GND	GND
P9	5	VCC[5V]	VCC	VCC
P9	11	GPIO[30]	FRC-21 pin	OUT
P9	12	GPIO[28]	FRC-22 pin	OUT
P9	13	GPIO[31]	FRC-19 pin	OUT
P9	14	GPIO[18]	FRC-20 pin	OUT

Test Plan:-

Test plans are prepared for each phase of testing. The initial test plan is created during the Project Planning phase. The initial test plan describes who performs which type of testing and when. Ideally master test plan covers all types of test i.e. from unit testing to production testing. The Lead Partner along with consortium partners is expected to submit the test plans to Utility for approval. Any changes made to the test plan during the project life cycle should be communicated to UTILITY for approval.

- Test plans contains following items:
- Roles and responsibilities of test team
- Approach to testing
- Function testing
- Security testing
- User Interface and reports testing
- Concurrency testing
- Performance and Load testing
- Test Scenarios along with entry and exit criteria
- Test specifications
- Suspension and resumption criteria

Sr No.	Test Case Name	Steps/Action	Expected Result	Actual Result	Remark
1	Checking connections between system and Beagle bone	Whether GPIO is assign or not	Beagle bone should be connected successfully.	Beagle bone was connected successfully.	Pass
2	Checking connection between Beagle bone and Stepper motor	Whether stepper motor working according to beagle bone or not	Stepper motor should be connected successfully	Stepper motor was connected successfully	Pass
3	Checking code	Whether code working right or not	Code should be correct	Code was correct	Pass
4	Checking Result	Whether result is generated or not	Result should be correct	Result was correct	Pass

Testing and Assurance:-

Testing and quality assurance in software development is more rigorous since each component has to be more reliable, if it is to be reused. A system is tested at various stages of development and deployment. For example, each component is tested as a unit for checking the correctness of its own code. Further, the component is tested with its dependent components. After final release of the entire set of components, system is tested for the correctness of system functionality. Finally the components are further tested in simulated production load for performance and load analysis. The Lead Partner along with consortium partners shall be responsible for the testing processes such as planning (includes preparing test plans and defining roles and their responsibilities), preparation (consists of preparing test specification, test environment and test data) and execution (includes testing at various levels like unit level, integration level, system level and production).

Conclusion

Hence, we have successfully developed the Project plan, SRS, Design document and Test Plan for one group-C assignment from embedded operating system or Concurrent and Distributed Programming.

FAQs:

1. What is Software Engineering?
2. Define Types of Testing()?
3. What are the concerns should consider while writing project plan?
4. What Design Documents and what it defines?
5. Define need of SRS?

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: B1

Title: Demonstration of Beeglebone black/ARM CPU Frequency

Aim: Write an application to and demonstrate the change in BeagleBoard/ ARM Cortex A5 /Microprocessor /CPU frequency or square wave of programmable frequency.

Prerequisites:

- Basics of ARM Processor
- Concept of Architecture of ARM and Frequency measures

Objectives:

- Ability to learn about ARM CPU frequency measures
- Ability to learn features of ARM

Theory:

What is BeagleBone Black?

BeagleBone Black is a low-cost, community-supported development platform for developers and hobbyists. Boot Linux in under 10 seconds and get started on development in less than 5 minutes with just a single USB cable.

Processor: AM335x 1GHz ARM® Cortex-A8

- 512MB DDR3 RAM
- 4GB 8-bit eMMC on-board flash storage
- 3D graphics accelerator
- NEON floating-point accelerator
- 2x PRU 32-bit microcontrollers

Connectivity:

- USB client for power & communications
- USB host
- Ethernet
- HDMI
- 2x 46 pin headers

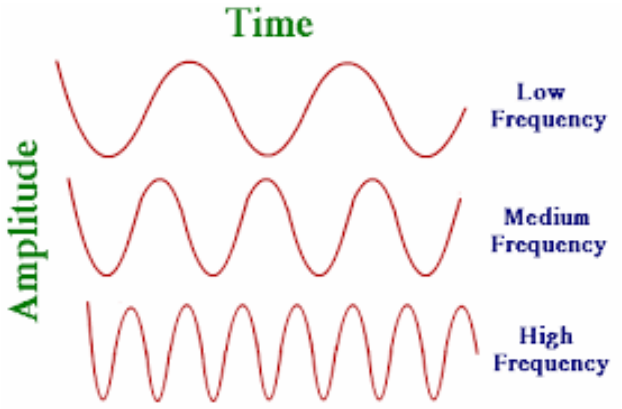
Software Compatibility:

- Debian
- Android
- Ubuntu

- Cloud9 IDE on Node.js w/ BoneScript library
- plus much more

Frequency: Frequency describes the number of waves that pass a fixed place in a given amount of time

Example: if the time it takes for a wave to pass is 1/2 second, the frequency is 2 per second. If it takes 1/100 of an hour, the frequency is 100 per hour. Usually frequency is measured in the **hertz** unit, named in honor of the 19th-century German physicist Heinrich Rudolf Hertz.



Conclusion:

Thus we have Writeen an application to and demonstrate the change in BeagleBoard/ ARM Cortex A5 /Microprocessor /CPU frequency or square wave of programmable frequency.

FAQs:

1. What is beagleboard black?
2. What are the features of beagle board balck?
3. What is the process to update the SW in the board?
4. How can boot from the microSD instead of the eMMC?
5. Explain the Device Tree?
6. Where can find Android for the BeagleBone Black?
7. How do get the BeagleBone Black Drivers to work on my Windows 8 computer?
8. What is the polarity of the DC power jack?
9. Define frequency?

10. What is the unit of frequency?
11. Draw diagram for different type of general frequency?
12. Define Amplitude?
13. Define Wavelength?
14. What is DC motor?
15. Explain working of DC Motor?
16. Write difference between AC and DC?
17. What is CRO?
18. List different types of embedded linux operating system?

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: B3

Title: Vedic Mathematics method to find square of 2-digit number

Aim: Vedic Mathematics method to find square of 2-digit number is used in a distributed programming. Use shared memory and distributed (multi-CPU) programming to complete the task.

Prerequisites:

Student should be able to

- Learn the concept distributed programming.
- Use the concepts of shared memory and distributed (multi-CPU) programming

Objectives:

Student should be able to

i) Define the shared memory and distributed concepts.

ii) Apply the distributed programming in different applications.

Theory:

Distributed Systems:

In distributed systems, there are two kinds of fundamental interprocess communication models: shared memory and message passing. From a programmer's perspective, shared memory computers, while easy to program, are difficult to build and aren't scalable to beyond a few processors. Message passing computers, while easy to build and scale, are difficult to program. In some sense, shared memory model and message passing model are equivalent. One of the solutions to parallel system communication is Distributed Shared Memory (DSM), where memory is physically distributed but logically shared. DSM appears as shared memory to the applications programmer, but relies on message passing between independent CPUs to access the global virtual address space.

Shared / Distributed Memory:

One of the first considerations in parallel programming is the relationship between the cores and the machine's memory. Each core in a multicore processor chip has coherent access to the same memory as the other cores in that chip. Many modern nodes have several sockets that allow several chips to be located on the same board. Each node in a parallel job will have one or more of the user's executables running on it. These executables will be called tasks. Shared-memory programming can be used within a task. A shared-memory program achieves its parallelism through threading.

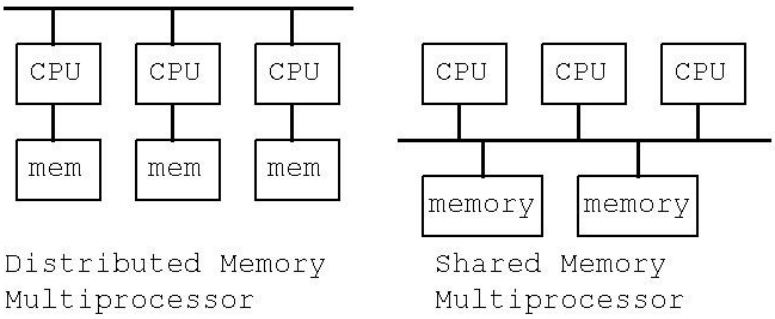


Fig: Shared / Distributed Memory:

In computer science, **distributed memory** refers to a multiple-processor computer system in which each processor has its own private memory. Computational tasks can only operate on local data, and if remote data is required, the computational task must communicate with one or more remote processors. In contrast, a shared memory multi processor offers a single memory space used by all processors. Processors do not have to be aware where data resides, except that there may be performance penalties, and that race conditions are to be avoided.

What is vedic mathematics?

Vedic Mathematics is the name given to the ancient system of Indian Mathematics which was rediscovered from the Vedas between 1911 and 1918 by Sri BharatiKrsnaTirthaji (1884-1960). According to his research all of mathematics is based on word-formulae. For example, 'Vertically and Crosswise` is one of these Sutras. These formulae describe the way the mind naturally works and are therefore a great help in directing the student to the appropriate method of solution.

Facilities:

Multicore CPU, CUDA/OpenCL or equivalent Open Source

Algorithm:

Vedic Mathematics method to find square of 2 digit numbers

To square any two digit number, follow these three simple steps:

Take each digit and square it. Then concatenate the two numbers. If one of the squares is less than 10, use 0 as a placeholder.

1. Take the two digits and multiply them. Then multiply by 2. Add a 0 on to the end.
2. Add your answers from steps 1 and 2.

Example:

Let the number is 56.

1. Start by taking the two digits and squaring them. 5^2 is 25, and 6^2 is 36. Concatenate them to get 2536.
2. Take the two digits and multiply them. 5 x 6 = 30.

3. Multiply this by 2. 30 x 2 = 60.
4. Add a zero on to the end to get 600.
5. Add the two numbers you got in the previous steps. 2536 + 600 = 3136.

And $56^2 = 3136$

Input:

2-digit number

Output:

Square of 2-digit number

Conclusion:

Distribute Programming concepts applied to implement Vedic Mathematics method to find square of 2-digit number.

Questions:

1. Explain any two examples of distributed systems.
2. Describe the limitations of distributed system.
3. Define distributed systems?
4. List three properties of distributed systems.
5. What is the concept of shared memory?
6. What is Vedic mathematics?
7. How to find square of two-digit number by Vedic Mathematics method?

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: B 4

Title: ODD-Even Sort algorithm

Aim: Implement a Parallel ODD-Even Sort algorithm using GPU or ARM equivalent

Prerequisites:

- Basics of CUDA programming

Objectives:

Students are able to learn parallel programming using GPU

Theory:

Parallelism on chip level is the hub for advancements in micro processor architectures for high performance computing. The core-processors, in personal computers, were not sufficient for high data-computation intensive tasks. As a result modular and specialized hardware in the form of sound cards or graphic accelerators are increasingly present in most personal computers.

Graphics cards or graphics processing units (GPU), introduced primarily for high-end gaming requiring high resolution. The GPU itself is a multi-core processor having support for thousands of threads running concurrently. GPU's are result of dozens of streaming processors with hundreds of core aligned in a particular way forming a single hardware unit.

Performance evaluation in GFLOPS (Giga Floating Point Operations per Second) shows that GPU's outperforms their CPU counterparts. For example: a high-end Core I7 processor (3.46 GHz) delivers up to a peak of 55.36 GFLOPs1.

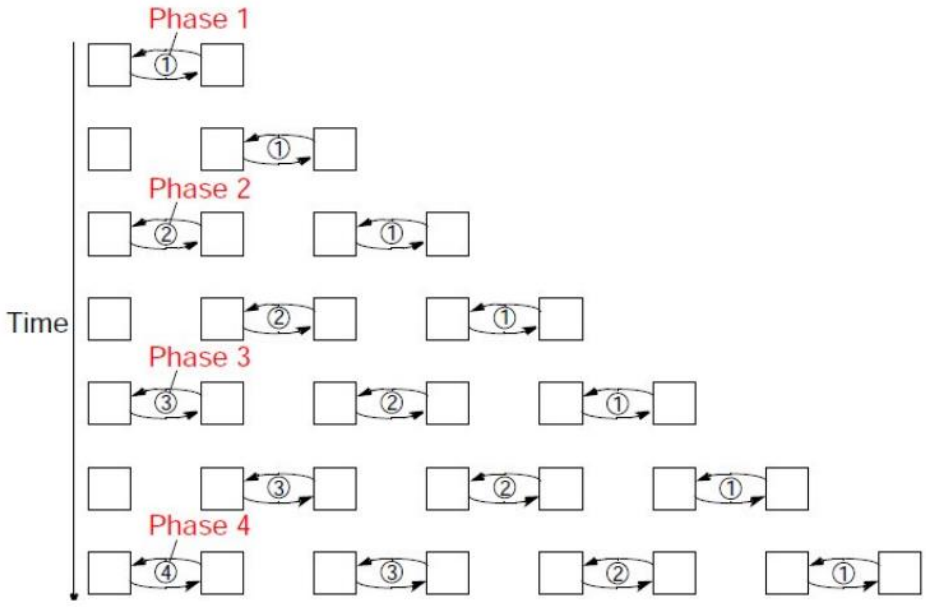
Parallel sorting algorithms

Sorting on GPU require transferring data from main memory to on-board GPU global memory. Although on-device bandwidth is in the range of 144Gb/s, thus only those sorting techniques are efficient which require minimum amount of synchronization because the PCI bandwidth is to the range of 2.5Gb/s. i.e., synchronization and memory transfers between CPU and GPU will affect system performance adversely. Compared to serial sorting algorithms, parallel algorithms are designed requiring high data independence between various elements for achieving better performance. Those techniques which involve large data dependency are categorized as sequential sorting algorithms.

Odd-Even Sort

The odd-even sort is a parallel sorting algorithm and is based on bubble-sort technique. Adjacent pairs of items in an array are exchanged if they are found to be out of order. What makes the technique distinct from bubble-sort is the technique of working on disjointed pairs, i.e., by using alternating pairs of odd-even and even-odd elements of the array. The technique works in multiple passes on a queue Q of size N. In each pass, elements at odd-numbered positions perform a comparison check based on

bubble-sort, after which elements at even- numbered positions do the same. The maximum number of iterations or passes for odd-even sort s N2. Total running time for this technique is O (log 2 N) .



Operates in two alternate phases:

Phase-even: Even processes exchange values with right neighbors.

Phase-odd: Odd processes exchange values with right neighbors.

Facilities:

Multicore CPU, CUDA/OpenCL or equivalent Open Source

Algorithm:

1. Start
2. Read numbers to sort
3. Sort by following method:

```
void ODD-EVEN-PAR(n)
{
        id = process label
        for (i= 1; i<= n; i++)
        {
                if (i is odd)
                compare-and-exchange-min(id+1);
                else
                compare-and-exchange-max(id-1);
                if (i is even)
                compare-and-exchange-min(id+1);
                else
                compare-and-exchange-max(id-1);
         }
}
```

4. Display result i.e. sorted numbers
5. Stop

Input:

Unsorted numbers

Output:

Sorted numbers

Conclusion:

Hence we have studied a Parallel ODD-Even Sort algorithm using GPU.

Questions:

8. What is complexity of Odd-Even Sort algorithm?
9. What is parallel programming?
10. Explain difference between CPU and GPU.
11. Explain parallel odd even sort algorithm.

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: B 5

Title: n-ary search algorithm

Aim: Implement n-ary search algorithm using OPENMP

Prerequisites

Concept of Linear and Binary search algorithm

Objectives:

Student should be able to learn the use of parallel programming for implementation of different algorithms.

Theory:

In k-ary search, we pick k-1 separators in each iteration, thereby dividing the search space into k partitions. As in binary search, the separators are chosen so that all partitions have equal size, i.e., they correspond to the i/k - quantiles of the dataset, where 1< i < k. The entire search process can be visualized using a k-ary search tree, in which nodes represent separators and children represent partitions. An example of such a tree is shown in Figure below.

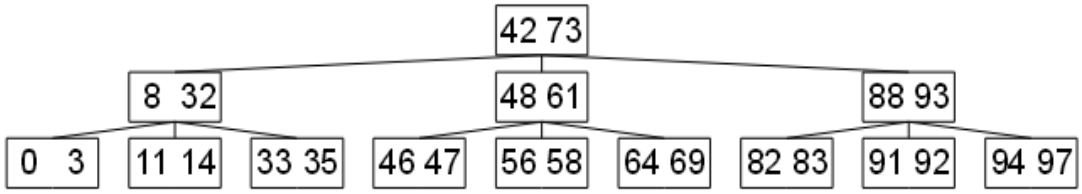


Fig: n-ary search for k=3

Suppose for the moment that the search tree is perfect, i.e., that every node including the root node has precisely k-1 entries, every internal node has k successors, and every leaf node has the same depth. For example, the tree shown in Figure above 3 is perfect. A perfect search tree can be constructed for any dataset of size $n = k^h - 1$ for some integer $h > 0$.

The search consists of the following steps, starting at the root:

S1: Load the k-1 separators of the current node into a register R.

S2: Terminate with success if the key is present in the register. Otherwise, terminate with failure when the current node is a leaf.

S3: Find the partition that encloses the search key.

S4: Move to the child node corresponding to that partition. Then, go to S1.

On a Sorted Array

We now show how to execute steps S1 and S4 of the above search algorithm on a sorted array A; steps S2 and S3 remain unmodified. As for binary search, we make use of two variables l (initialized to 1) and r (initialized to n) that represent the left and right end of the remaining search space. Then, step S1 takes l and r as input and loads the separators into R. Step S4 takes m as an additional input and updates l and r to point to the next partition (given m).

Step S1 consists of two parts:

(a) calculate the indexes of the separators

(b) load them into R. Substep (a) is required because the k-1 separators are stored in non-contiguous memory locations,

With p = r – l + 1 denoting the number of elements in the current partition, we find that the separators are stored at positions l + id [(p + 1) / k] for 1

Facilities:

Latest version of 64 Bit Operating Systems, OpenMP

Algorithm:

1. A node generally has **m-1** keys and **m** children. Each node has alternating sub-tree pointers and keys: sub-tree | key | sub-tree | key | ... | key | sub_tree
2. All keys in a sub-tree to the left of a key are smaller than it.
3. All keys in the node *between* two keys are between those two keys.
4. All keys in a sub-tree to the right of a key are greater than it.

This is the "standard" recursive part of the algorithm

Input:

A sorted array.

Output:

Element to find.

Conclusion:

We learned the concept of parallel programming using OpenMP.

Questions:

1. What is nary search algorithm?
2. What is OpenMP? How it is used for parallel programming?
3. Explain the various architectural schemes used in parallel processor.
4. What is the need for parallel processing techniques?
5. Explain various parallel programming models
6. Explain performance analysis theorems for parallel computers.

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: B 6

Title: Prims algorithm using OPENMP

Aim: Implement concurrent prims algorithm using OPENMP

Prerequisites:

- Concept of minimum spanning tree and Prims algorithm
- Basics of concurrent programming

Objectives:

Student should be able to learn the use of concurrent programming for implementation of different algorithms.

Theory:

OpenMP :

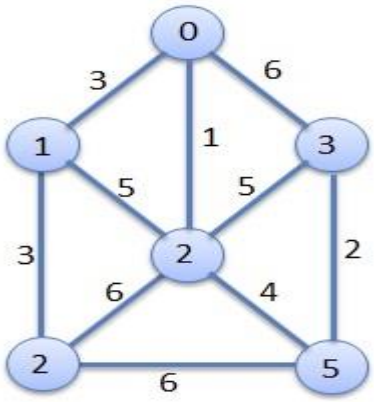
OpenMP consists of a set of compiler #pragmas that control how the program works. The pragmas are designed so that even if the compiler does not support them, the program will still yield correct behavior, but without any parallelism.

Prim's Algorithm :

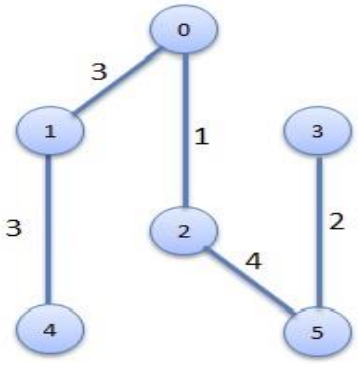
Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph. It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. This algorithm is directly based on the MST (minimum spanning tree) property.

Example :

A Simple Weighted Graph



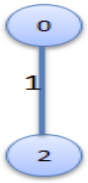
Minimum-Cost Spanning Tree



Procedure for finding Minimum Spanning Tree

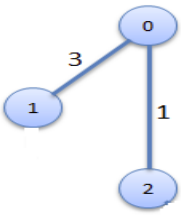
Step 1 :

No. of Nodes	0	1	2	3	4	5
Distance	0	3	1	6	∞	∞
Distance From		0	0	0		



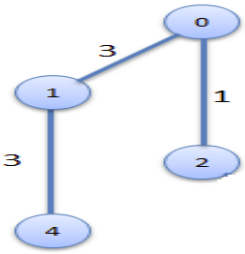
Step 2 :

No. of Nodes	0	1	2	3	4	5
Distance	0	3	0	5	6	4
Distance From	0		2	2	2	



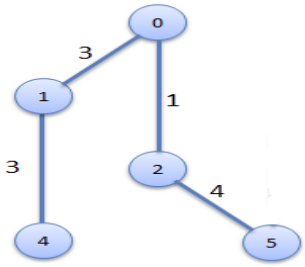
Step 3 :

No. of Nodes	0	1	2	3	4	5
Distance	0	0	0	5	3	4
Distance From			2	1	2	



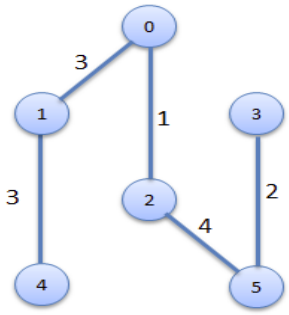
Step 4 :

No. of Nodes	0	1	2	3	4	5
Distance	0	0	0	5	0	4
Distance From				2	2	



Step 5 :

No. of Nodes	0	1	2	3	4	5
Distance	0	0	0	3	0	0
Distance From				2	2	



Minimum Cost = 1+2+3+3+4 = 13

Facilities:

Latest version of 64 Bit Operating Systems, OpenMP

Algorithm:

1) Create a set *mstSet* that keeps track of vertices already included in MST.

2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

3) While mstSet doesn't include all vertices

….a) Pick a vertex *u* which is not there in *mstSet* and has minimum key value.

….b) Include *u* to mstSet.

….c) Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u*-*v* is less than the previous key value of *v*, update the key value as weight of *u*-*v*

Input:

1. Weighted Graph with number of lanes i. e. edges and number of houses i. e. vertices
2. Starting vertex.

Output:

Minimum spanning tree of given weighted graph.

Conclusion:

Thus, we have implemented prim's algorithm using OpenMP.

Questions:

1. What is minimum spanning tree of a graph?
2. What is concurrent programming?
3. Explain difference between concurrent, parallel and distributed programming.
4. Differentiate between Prim's and Kruskal's Algorithm.

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: B7

Title: *nxn* matrix parallel multiplication using CUDA/OpenCL GPU

Aim: Implement *nxn* matrix parallel multiplication using CUDA/OpenCL GPU, use shared memory.

Prerequisites:

- Concept of matrix multiplication.
- Basics of CUDA programming

Objectives:

Student should be able to learn parallel programming, CUDA architecture and CUDA processing flow

Theory:

A straightforward matrix multiplication example that illustrates the basic features of memory and thread management in CUDA programs

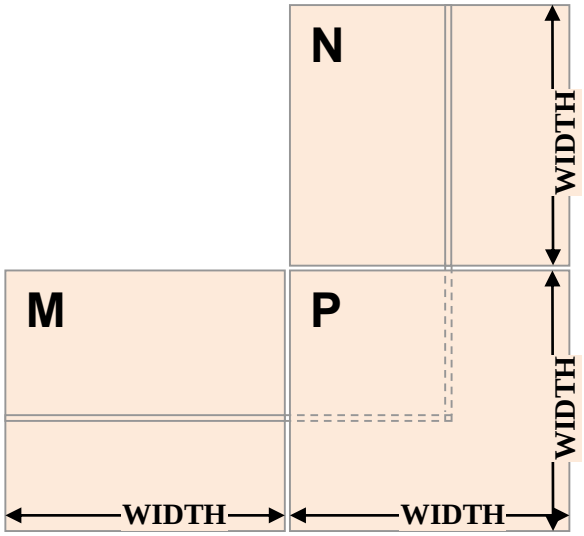
- Leave shared memory usage until later
- Local, register usage
- Thread ID usage

Memory data transfer API between host and device

- *P = M * N of size WIDTH x WIDTH*

Without tiling:

- One thread handles one element of P
- M and N are loaded WIDTH times from global memory



Matrix Multiplication steps

8. **Matrix Data Transfers**
9. **Simple Host Code in C**
10. **Host-side Main Program Code**
11. **Device-side Kernel Function**
12. **Some Loose Ends**

Step 1: Matrix Data Transfers

```
// Allocate the device memory where we will copy M to
Matrix Md;
Md.width  = WIDTH;
Md.height = WIDTH;
Md.pitch  = WIDTH;
int size = WIDTH * WIDTH * sizeof(float);
cudaMalloc((void**)&Md.elements, size);
// Copy M from the host to the device
cudaMemcpy(Md.elements, M.elements, size, cudaMemcpyHostToDevice);
// Read M from the device to the host into P
cudaMemcpy(P.elements, Md.elements, size, cudaMemcpyDeviceToHost);
...
// Free device memory
cudaFree(Md.elements);
```

Step 2: Simple Host Code in C

```
// Matrix multiplication on the (CPU) host in double precision
// for simplicity, we will assume that all dimensions are equal
void MatrixMulOnHost(const Matrix M, const Matrix N, Matrix P)
{
    for (int i = 0; i < M.height; ++i)
        for (int j = 0; j < N.width; ++j) {
            double sum = 0;
            for (int k = 0; k < M.width; ++k) {
                double a = M.elements[i * M.width + k];
                double b = N.elements[k * N.width + j];
                sum += a * b;
            }
            P.elements[i * N.width + j] = sum;
        }
```

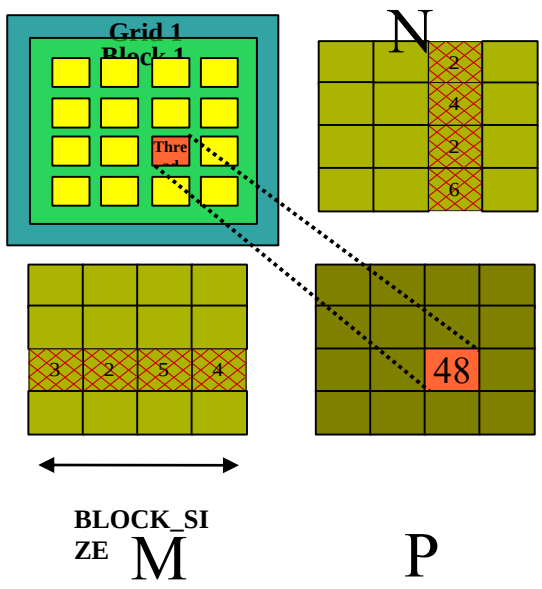
Multiply Using One Thread Block

- One Block of threads compute matrix P

- Each thread computes one element of P

- Each thread
  - Loads a row of matrix M
  - Loads a column of matrix N
  - Perform one multiply and addition for each pair of M and N elements
  - Compute to off-chip memory access ratio close to 1:1 (not very high)

- Size of matrix limited by the number of threads allowed in a thread block



Step 3: Host-side Main Program Code

```
int main(void) {
// Allocate and initialize the matrices
    Matrix  M  = AllocateMatrix(WIDTH, WIDTH, 1);
    Matrix  N  = AllocateMatrix(WIDTH, WIDTH, 1);
    Matrix  P  = AllocateMatrix(WIDTH, WIDTH, 0);
// M * N on the device
    MatrixMulOnDevice(M, N, P);
    FreeMatrix(M);
    FreeMatrix(N);
    FreeMatrix(P);
return 0;
}
```

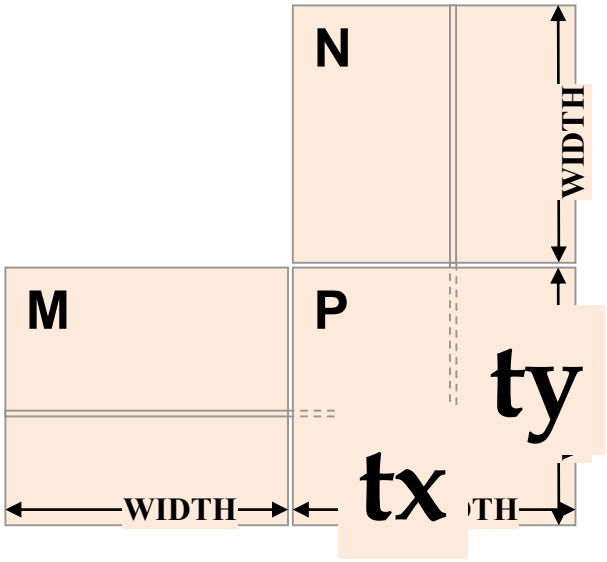
Host-side code

```
// Matrix multiplication on the device
void MatrixMulOnDevice(const Matrix M, const Matrix N, Matrix P)
{
```

```
    // Load M and N to the device
    Matrix Md = AllocateDeviceMatrix(M);
    CopyToDeviceMatrix(Md, M);
    Matrix Nd = AllocateDeviceMatrix(N);
    CopyToDeviceMatrix(Nd, N);
    // Allocate P on the device
    Matrix Pd = AllocateDeviceMatrix(P);
    CopyToDeviceMatrix(Pd, P); // Clear memory
    // Setup the execution configuration
    dim3 dimBlock(WIDTH, WIDTH);
    dim3 dimGrid(1, 1);
    // Launch the device computation threads!
    MatrixMulKernel<<<dimGrid, dimBlock>>>(Md, Nd, Pd);
    // Read P from the device
    CopyFromDeviceMatrix(P, Pd);
    // Free device matrices
    FreeDeviceMatrix(Md);
    FreeDeviceMatrix(Nd);
    FreeDeviceMatrix(Pd);
}
```

Step 4: Device-side Kernel Function

```
// Matrix multiplication kernel – thread specification
__global__ void MatrixMulKernel(Matrix M, Matrix N, Matrix P)
{
    // 2D Thread ID
    int tx = threadIdx.x;
    int ty = threadIdx.y;
    // Pvalue is used to store the element of the matrix
    // that is computed by the thread
    float Pvalue = 0;
for (int k = 0; k < M.width; ++k)
    {
         float Melement = M.elements[ty * M.pitch + k];
         float Nelement = Nd.elements[k * N.pitch + tx];
         Pvalue += Melement * Nelement;
    }
    // Write the matrix to device memory;
    // each thread writes one element
    P.elements[ty * P.pitch + tx] = Pvalue;
}
```



Step 5: Some Loose Ends

- Free allocated CUDA memory

Facilities:

Latest version of 64 Bit Operating Systems, CUDA enabled NVIDIA Graphics card

Algorithm:

1. Start
2. Read Matrix and transfer Data to Device
3. Perform Matrix Multiplication at Device side by kernel function
4. Return results from Device-side to Host
5. Display Results
6. Stop

Input:

Two matrices

Output:

Multiplication of two matrix

Conclusion:

We learned parallel programming with the help of CUDA architecture.

Questions:

1. What is CUDA?
2. Explain Processing flow of CUDA programming.
3. Explain advantages and limitations of CUDA.
4. Make the comparison between GPU and CPU.
5. Explain various alternatives to CUDA.
6. Explain CUDA hardware architecture in detail.

Experiment No:

Title: _____

Roll No: _ _ _ Batch: _ _ _

Date of Performance: _ _/_ _/_ _ _ _

Date of Assessment::_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: B8

Title: Network based Client-Server program.

Aim: Develop a network based application by setting IP address on BeagleBoard/ ARM Cortex A5.

Prerequisites:

- Concept of Socket Programming
- Basics of Computer Network

Objectives:

- Ability to learn significance of network based programming with ARM based Processor.

Theory:

Introduction:

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard UNIX file descriptors. In UNIX, every I/O actions are done by writing or reading to a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.
To a programmer a socket looks and behaves much like a low level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes. The difference between sockets and normal file descriptors occurs in the creation of a socket and through a variety of special operations to control a socket.
Sockets were first introduced in 2.1BSD and subsequently refined into their current form with 4.2BSD. The sockets feature is now available with most current UNIX system releases.

Socket usage:

A Unix Socket is used in a client server application framework. A server is a process which does some function on request from a client. Most of the application level protocols like FTP, SMTP and POP3 make use of Sockets to establish connection between client and server and then for exchanging data.

Socket types:

There are four types of sockets available to the users. The first two are most commonly used and last two are rarely used.
Processes are presumed to communicate only between sockets of the same type but there is no restriction that prevents communication between sockets of different types.

1) **Stream Sockets:**

Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A,B,C", they will arrive in the same order - "A,B,C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.

2) **Datagram Sockets:**

Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets - you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

3) **Raw Sockets:**

It provides users access to the underlying communication protocols which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more esoteric facilities of an existing protocol.

4) **Sequenced Packet Sockets:**

They are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications. Sequenced-packet sockets allow the user to manipulate the Sequence Packet Protocol (SPP) or Internet Datagram Protocol (IDP) headers on a packet or a group of packets either by writing a prototype header along with whatever data is to be sent, or by specifying a default header to be used with all outgoing data, and allows the user to receive the headers on incoming packets.

Client server model:

Most of the Net Applications use the Client Server architecture. These terms refer to the two processes or two applications which will be communicating with each other to exchange some information. One of the two processes acts as a client process and another process acts as a server.

Client process

This is the process which typically makes a request for information. After getting the response this process may terminate or may do some other processing.
For example: Internet Browser works as a client application which sends a request to Web Server to get one HTML web page.

Sever process
This is the process which takes a request from the clients. After getting a request from the client, this process will do required processing and will gather requested information and will send it to the requestor client. Once done, it becomes ready to serve another client. Server processes are always alert and ready to serve incoming requests.
For example: Web Server keeps waiting for requests from Internet Browsers and as soon as it gets any request from a browser, it picks up a requested HTML page and sends it back to that Browser.
Notice that the client needs to know of the existence and the address of the server, but the server does not need to know the address or even the existence of the client prior to the connection being established. Once a connection is established, both sides can send and receive information.

2-tier and 3-tier architectures:

There are two types of client server architectures:

- **2-tier architectures:** In this architecture, client directly interacts with the server. This type of architecture may have some security holes and performance problems. Internet Explorer and Web Server works on two tier architecture. Here security problems are resolved using Secure Socket Layer (SSL).

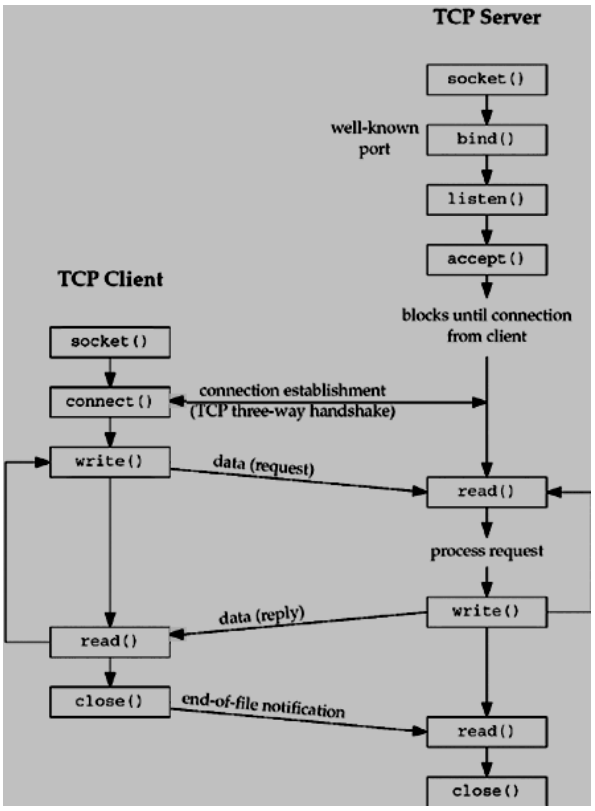
- **3-tier architectures:** In this architecture, one more software sits in between client and server. This middle software is called middleware. Middleware are used to perform all the security checks and load balancing in case of heavy load. A middleware takes all requests from the client and after doing required authentication it passes that request to the server. Then server does required processing and sends response back to the middleware and finally middleware passes this response back to the client. If you want to implement 3-tier architecture then you can keep any middle ware like Web Logic or WebSphere software in between your Web Server and Web Browsers.

Types of Server:
There are two types of servers you can have:

i) **Iterative Server:** This is the simplest form of server where a server process serves one client and after completing first request then it takes request from another client. Meanwhile another client keeps waiting.

ii) **Concurrent Servers:** This type of server runs multiple concurrent processes to serve many requests at a time. Because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to *fork* a child process to handle each client separately.

Client server interaction:
Following figure 1 show complete Client and Server interaction:



Socket structure :
There are various structures which are used in Unix Socket Programming to hold information about the address and port and other information. Most socket functions require a pointer to a socket address structure as an argument. Structures defined in this tutorial are related to Internet Protocol Family.
The first structure is struct *sockaddr* that holds socket information:

```
struct sockaddr{
        unsigned short  sa_family;
        char            sa_data[14];
};
```

This is a generic socket address structure which will be passed in most of the socket function calls. Here is the description of the member fields:

Attribute	Values	Description
sa_family	AF_INET AF_UNIX AF_NS AF_IMPLINK	This represents an address family. In most of the Internet based applications we use AF_INET.

sa_data	Protocol Specific Address	The content of the 14 bytes of protocol specific address are interpreted according to the type of address. For the Internet family we will use port number IP address which is represented by*sockaddr_in* structure defined below.

Second structure that helps you to reference to the socket's elements is as follows:

```
struct sockaddr_in {
        short int          sin_family;
        unsigned short int   sin_port;
        struct in_addr     sin_addr;
        unsigned char      sin_zero[8];
};
```

Here is the description of the member fields:

Attribute	**Values**	**Description**
sa_family	AF_INET AF_UNIX AF_NS AF_IMPLINK	This represents an address family. In most of the Internet based applications we use AF_INET.
sin_port	Service Port	A 16 bit port number in Network Byte Order.
sin_addr	IP Address	A 32 bit IP address in Network Byte Order.
sin_zero	Not Used	You just set this value to NULL as this is not being used.

Socket Core Functions/System Calls:

1.The socket system call

To perform network I/O, the first thing a process must do is call the socket function, specifying the type of communication protocol desired and protocol family etc.
Signature: int socket (int family, int type, int protocol);

2.The connect system call

The *connect* function is used by a TCP client to establish a connection with a TCP server.

Signature: int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);

3.The bind system call

The *bind* function assigns a local protocol address to a socket. With the Internet protocols, the protocol address is the combination of either a 32-bit IPv4 address or a 128-bit IPv6 address, along with a 16-bit TCP or UDP port number. This function is called by TCP server only.
Signature: int bind(int sockfd, struct sockaddr *my_addr,int addrlen);

4.The listen system call

The *listen* function is called only by a TCP server and it performs two actions:
The listen function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.
The second argument to this function specifies the maximum number of connections the kernel should queue for this socket.
Signature: int listen(int sockfd, int backlog);

5. The accept system call:

The accept function is called by a TCP server to return the next completed connection from the front of the completed queue. Following is the signature of the call.
Signature: int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);

6.The send system call

The *send* function is used to send data over stream sockets or CONNECTED datagram sockets. If you want to send data over UNCONNECTED datagram sockets you must use sendto() function.You can use *write()* system call to send the data.
Signature: int send(int sockfd, const void *msg, int len, int flags);

8.The recv system call

The *recv* function is used to receive data over stream sockets or CONNECTED datagram sockets. If you want to receive data over UNCONNECTED datagram sockets you must use recvfrom().You can use *read()* system call to read the data.
Signature: int recv(int sockfd, void *buf, int len, unsigned int flags);

9.The sendto system call

The *sendto* function is used to send data over UNCONNECTED datagram sockets. Put simply, when you use scoket type as SOCK_DGRAM. This call returns the number of bytes sent otherwise it will return -1 on error.
Signature: int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);

10.The recvfrom system call

The *recvfrom* function is used to receive data from UNCONNECTED datagram sockets. Put simply, when you use scoket type as SOCK_DGRAM. This call returns the number of bytes read into the buffer otherwise it will return -1 on error.
Signature: int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);

11.The close system call
The *close* function is used to close the communication between client and server. This call returns 0 on success otherwise it will return -1 on error.
Signature: int close(int sockfd);

12.The shutdown system call

The *shutdown* function is used to gracefully close the communication between client and server. This function gives more control in caomparision of *close* function. This call returns 0 on success otherwise it will return -1 on error.
Signature: int shutdown(int sockfd, int how);

Using Minicom

1. Start minicom on your host machine in configuration mode. As root:
2. # minicom -s
3. A menu of configuration should appear. Use the Down-arrow key to scroll down and select the Serial port setup option, and press Enter.
4. Verify that the listed serial port is the same one that is connected to the target board. If it is not, press A, and enter the correct device. This is /dev/ttyACM0 on most Linux distributions and press Enter.
5. Set Hardware flow control to No using the F key.
6. Set Software flow control to YES using the G key.
7. Press Enter to return to the main configuration menu, and then press Esc to exit this menu.
8. Reset the board, and wait for a moment. If you do not see output from the board, press Enter several times until you see the prompt. If you do not see any output from the board, and have verified that the serial terminal connection is setup correctly, contact your board vendor.
9. Save setup as default & exit
10. Target will boot & will ask login&password
11. After login switch to super user ($ su)
12. Create folder for your project
13. Create source file in the folder, use vim Text Editor create source file
14. Compile source file (use g++ compiler)
15. Execute output file
16. Verify output

Conclusion:

Hence we have understood the concept of developing a network based application by setting IP address on BeagleBoard/ ARM Cortex A5 by implementing Client-Server Program.

FAQs:

1. What is default static IP address of Beagle Black Bone?
2. How to change IP address of Beagle Black Bone?
3. Enlist the steps to set IP address of Beagle Black Bone.
4. How network based application like CHAT works on Linux operating System.
5. How network based application like File transfer works on Linux operating System.
6. Explain different types of Communication.
7. Enlist the protocols used for Communication Purpose.
8. Enlist different commands which are used to execute program with Beagle Black Bone.
9. Explain the different applications of Beagle Black Bone.

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: B 9

Title: Multi-threading application for echo server using socket programming

Aim: Implement a Multi-threading application for echo server using socket programming in JAVA

Prerequisites:

Basics of Java, Multithreading concept and basic knowledge of protocols

Objectives:

i) To study socket programming in java

ii) To study Multi-threading concept in java.

Theory:

Introduction to socket

A socket is the one end-point of a two-way communication link between two programs running over the network. Running over the network means that the programs run on different computers, usually referred as the local and the remote computers. However one can run the two programs on the same computer. Such communicating programs constitutes a client/server application. The server implements a dedicated logic, called **service**. The clients connect to the server to get served, for example, to obtain some data or to ask for the computation of some data. Different client/server applications implement different kind of services.

Socket Programming :

Socket programming is useful for building client-server applications.

Socket:

The **java.net.ServerSocket** class is used by server applications to obtain a port and listen for client requests.

ServerSocket(int port)	Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
getLocalPort()	Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
Socket accept()	Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely

Socket(InetAddress host, int port)	This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.

Creating with threads in Java:

1. By Implementing Runnable Interface

If your class is intended to be executed as a thread then you can achieve this by implementing **Runnable** interface. You will need to follow three basic steps:

Step 1: As a first step you need to implement a run() method provided by Runnable interface. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run() method:

```
public void run( )
```

Step 2: At second step you will instantiate a Thread object using the following constructor:

```
Thread(Runnable threadObj, String threadName);
```

Where, *threadObj* is an instance of a class that implements the **Runnable** interface and **threadName** is the name given to the new thread.

Step 3: Once Thread object is created, you can start it by calling start( ) method, which executes a call to run( ) method. Following is simple syntax of start() method:

```
void start( );
```

2. By Implementing Extending Thread Class

The second way to create a thread is to create a new class that extends **Thread** class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1: You will need to override run( ) method available in Thread class. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run() method:

```
public void run( )
```

Step 2: Once Thread object is created, you can start it by calling start( ) method, which executes a call to run( ) method. Following is simple syntax of start() method:

```
void start( );
```

Client :

- Creates a socket with the same port number :
  Socket clientSocket = new Socket("localhost", 6789);
- Gets input/output streams.
- Exchanges information with the server.
- Clean up.

ECHO Server :

An "echo server" is a server that does nothing more than sending back whatever is sent to it. Hence the name : echo What can you use it for ? Whatever you feel like. Practical applications could be network and connectivity testing and troubleshooting. Assume you've build a rather complex network with VLANs and subnets, and really strick firewalls between those subnets, and you're beginning to wonder if a client on one segment of the network will still be able to connect to your web server, database server, ... on some other segment. A ping or a traceroute will establish if the server (IP address) can be reached but does not tell you if an application will be able to connect to the desired port on the server and whether a reply from the server will be able to reach the client again.

This "echo server" can be set up to listen on any desired (tcp) port to simulate whatever application you want to run (eg web server = port 80, Microsoft SQL Server = port 1433, etc). From the client machine, you can then telnet to this port. When a telnet connection has been established, everything you type will be echoed back to your screen, indicating that the telnet client and the echo server can talk to each other : you've established connectivity at the application level.

In a similar way, you can use this echo server to troubleshoot networks, test a firewall (eg "if I have a server listening on port 123, wil my firewall allow connections to it ?) and so on.

Echo server :

1. The client reads a line from its standard input and writes that line to the server.
2. The server reads a line from its network input and echoes the line back to the client over the network.
3. The client reads the echoed line from the network and prints it on its standard output.

Multithreaded Server Advantages :

The advantages of a multithreaded server compared to a single threaded server are summed up below:

- Less time is spent outside the accept() call.
- Long running client requests do not block the whole server.
- In a single threaded server long running requests may make the server unresponsive for a long period. This is not true for a multithreaded server, unless the long-running request takes up all CPU time and/or network bandwidth.

Facilities:

Latest version of 64 Bit Operating Systems, Eclipse 64-bit Platform, jdk 1.6

Algorithm:

Server

1. Create a server socket and bind it to port.

2. Listen for new connection and when a connection arrives, accept it.

3. Read Client's message and display it

4. Get a message from user and send it to client

5. Repeat steps 3-4 until the client sends "end"

6. Close all streams

7. Close the server and client socket

8. Stop

Client

1. Create a client socket and connect it to the server's port number

2. Get a message from user and send it to server

3. Read server's response and display it

4. Repeat steps 2-3 until chat is terminated with "end" message

5. Close all input/output streams

6. Close the client socket

7. Stop

Input:

Message from client

Output:

Message from server

Conclusion:

Java sockets API (Socket and ServerSocket classes) is a powerful and flexible interface for network programming of client/server applications. On the other hand, Java threads are another powerful programming framework for client/server applications. Multi-threading simplifies the implementation of complex client/server applications.

Questions:

1. What is Echo server?
2. What are the methods to implement thread in java?
3. List different socket methods under java.
4. What is mean by multithreading?

Experiment No:

Title: _____

Roll No:_ _ _ Batch:_ _ _

Date of Performance:_ _/_ _/_ _ _ _

Date of Assessment: :_ _/_ _/_ _ _ _

Particulars	Marks
Attendance (05)	
Journal (05)	
Performance (05)	
Understanding(05)	
Total (20)	
Signature of Staff Member	

Experiment No: C1

Title: Robotics Stepper Motor Application

Aim: Develop Robotics (stepper motor) Application using Beagle Board.

Prerequisites:

- Basics working principles of Stepper Motor
- Fundamentals of Python Programming

Objectives:

- To develop a Robotics Stepper Motor Application.
- Implementation of Program using Python

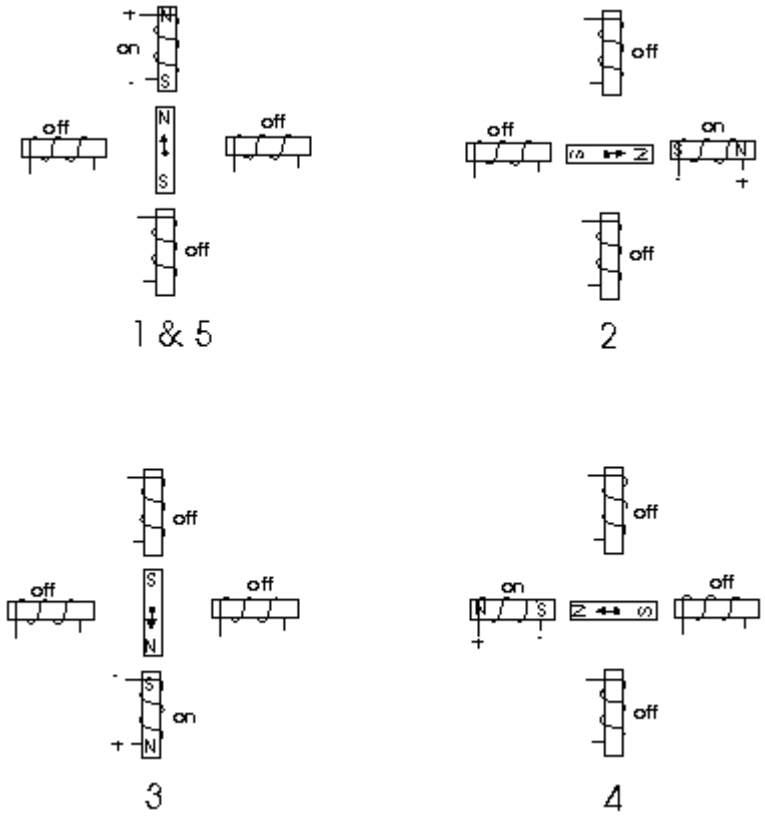
Theory:

Introduction

A step motor can be viewed as a synchronous AC motor with the number of poles (on both rotor and stator) increased, taking care that they have no common denominator. Additionally, soft magnetic material with many teeth on the rotor and stator cheaply multiplies the number of poles (reluctance motor). Modern steppers are of hybrid design, having both permanent magnets and soft iron cores.

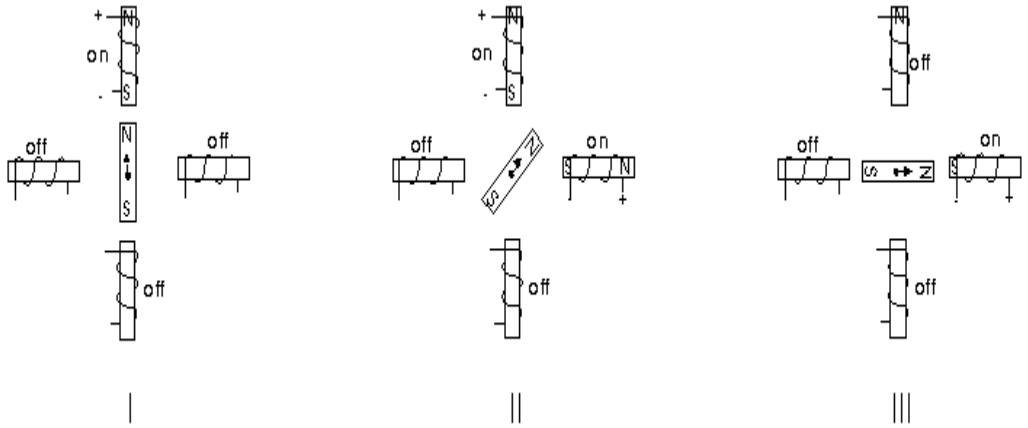
Working of Stepper Motor:

Stepper motors consist of a permanent magnetic rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator. Following illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.



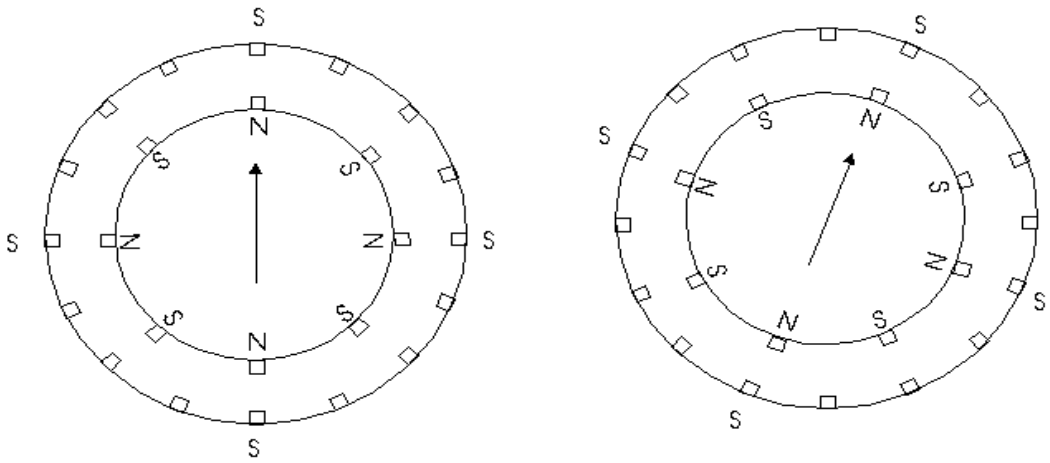
In the above example, we used a motor with a resolution of 90 degrees or demonstration purposes. In reality, this would not be a very practical motor for most applications. The average stepper motor's resolution -- the amount of degrees rotated per pulse -- is much higher than this. For example, a motor with a resolution of 5 degrees would move its rotor 5 degrees per step, thereby requiring 72 pulses (steps) to complete a full 360 degree rotation.

You may double the resolution of some motors by a process known as "half-stepping". Instead of switching the next electromagnet in the rotation on one at a time, with half stepping you turn on both electromagnets, causing an equal attraction between, thereby doubling the resolution. As you can see in following figure, in the first position only the upper electromagnet is active, and the rotor is drawn completely to it. In position 2, both the top and right electromagnets are active, causing the rotor to position itself between the two active poles. Finally, in position 3, the top magnet is deactivated and the rotor is drawn all the way right. This process can then be repeated for the entire rotation.



There are several types of stepper motors. 4-wire stepper motors contain only two electromagnets; however the operation is more complicated than those with three or four magnets, because the driving circuit must be able to reverse the current after each step. For our purposes, we will be using a 6-wire motor.

Unlike our example motors which rotated 90 degrees per step, real-world motors employ a series of mini-poles on the stator and rotor to increase resolution. Although this may seem to add more complexity to the process of driving the motors, the operation is identical to the simple 90 degree motor we used in our example. An example of a multipole motor can be seen in following figure. In position 1, the north pole of the rotor's permanent magnet is aligned with the south pole of the stator's electromagnet. Note that multiple positions are aligned at once. In position 2, the upper electromagnet is deactivated and the next one to its immediate left is activated, causing the rotor to rotate a precise amount of degrees. In this example, after eight steps the sequence repeats.



Conclusion:

Thus we have developed Robotics (stepper motor) Application using Beagle Board.

FAQs:

1. What is stepper motor?
2. Which are the various types of stepper motor?
3. What is difference between stepper motor and regular motor?
4. What is difference between stepper motor and servo motor?
5. What do you mean by unipolar stepper motor?
6. What do you mean by bipolar stepper motor?
7. What is Permanent magnet stepper?
8. What is Hybrid synchronous stepper?
9. What is Variable reluctance stepper?
10. What is Lavet type stepping motor?
11. What is the use of Stepper motor controllers?
12. Explain concept of fullstepping.
13. Explain concept of microstepping.
14. Explain concept of halfstepping.
15. What is robotics?
16. What is the definition of a 'robot'?
17. When did robots, as we know them today, come into existence?
18. Which are the various applications of Robotics?
19. How many stepper motors are required in Robotic Arm?
20. Can Robotic arm work in water or ice?