**Group A-1**

**Problem Statement**: Study of Raspberry-Pi, Beagle board, Arduino and other micro controller (History & Elevation)

**Apparatus :** Raspberry Pi, Beagle Board and Arduino

**Theory:**

**Introduction to Embedded System**

An **embedded system** is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is *embedded* as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors are manufactured as components of embedded systems.

Examples of properties of typical embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available. For example, intelligent techniques can be designed to manage power consumption of embedded systems.

Modern embedded systems are often based on microcontrollers (i.e. CPU's with integrated memory or peripheral interfaces),but ordinary microprocessors (using external chips for memory and peripheral interface circuits) are also common, especially in more-complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP).

**Why we use embedded system?**

The **uses** of **embedded systems** are virtually limitless, because every day new products are introduced to the market that utilize **embedded** computers in novel ways. In recent years, hardware such as microprocessors, microcontrollers, and FPGA chips have become much cheaper.

An **embedded system** is a computer **system** with a dedicated **function** within a larger mechanical or electrical **system**, often with real-time computing constraints. It is **embedded** as part of a complete device often including hardware and mechanical parts.

**Raspberry Pi**

The **Raspberry Pi** is a series of small [single-board computers](https://en.wikipedia.org/wiki/Single-board_computer) developed in the [United Kingdom](https://en.wikipedia.org/wiki/United_Kingdom) by the [Raspberry Pi Foundation](https://en.wikipedia.org/wiki/Raspberry_Pi_Foundation) to promote the teaching of basic [computer science](https://en.wikipedia.org/wiki/Computer_science) in schools and in [developing countries](https://en.wikipedia.org/wiki/Developing_countries).The original model became far more popular than anticipated, selling outside its [target market](https://en.wikipedia.org/wiki/Target_market) for uses such as [robotics](https://en.wikipedia.org/wiki/Robotics). It does not include peripherals (such as [keyboards](https://en.wikipedia.org/wiki/Keyboard_%28computing%29), [mice](https://en.wikipedia.org/wiki/Mouse_%28computing%29) and [cases](https://en.wikipedia.org/wiki/Computer_case)). However, some accessories have been included in several official and unofficial bundles.

According to the Raspberry Pi Foundation, over 5 million Raspberry Pis were sold by February 2015, making it the best-selling [British computer](https://en.wikipedia.org/wiki/British_computer). By November 2016 they had sold 11 million units, and 12.5m by March 2017, making it the third best-selling "general purpose computer". In July 2017, sales reached nearly 15 million.

**Evolution of Raspberry Pi :**

The first generation (Raspberry Pi 1 Model B) was released in February 2012, followed by the simpler and cheaper Model A. In 2014, the Foundation released a board with an improved design, Raspberry Pi 1 Model B+. These boards are approximately credit-card sized and represent the standard mainline form-factor. Improved A+ and B+ models were released a year later. A “Compute Model” was released in April 2014 for embedded applications. The Raspberry Pi 2 which added more RAM was released in February 2015.

**A Raspberry Pi Zero** with smaller size and reduced I/O and general-purpose I/O (GPIO) capabilities was released in November 2015 for US$5. Raspberry Pi 3 Model B was released in February 2016 and has on-board Wi-Fi, Bluetooth and USB boot capabilities. By 2017, it became the newest mainline Raspberry Pi. On 28 February 2017, the Raspberry Pi Zero W was launched, a version of the Zero with Wi-Fi and Bluetooth capabilities, for US$10.

**Raspberry Pi 2 V1.1** included a quad-core Cortex-A7 CPU running at 900 MHz and 1 GB RAM. It was described as 4–6 times more powerful than its predecessor. The GPU was identical to the original. In parallelized benchmarks, the Raspberry Pi 2 V1.1 could be up to 14 times faster than a Raspberry Pi 1Model B+.

**The Raspberry Pi 3,** with a quad-core ARM Cortex-A53 processor, is described as 10 times the performance of a Raspberry Pi 1. This was suggested to be highly dependent upon task threading and instructions set use. Benchmarks showed the Raspberry Pi 3 to be approximately 80% faster than the Raspberry Pi 2 in parallelized tasks.

In 2006, early concepts of the Raspberry Pi were based on the [Atmel](https://en.wikipedia.org/wiki/Atmel) [ATmega](https://en.wikipedia.org/wiki/ATmega)644 microcontroller. Its schematics and [PCB](https://en.wikipedia.org/wiki/Printed_circuit_board) layout are publicly available. Foundation [trustee](https://en.wikipedia.org/wiki/Trustee) [Eben Upton](https://en.wikipedia.org/wiki/Eben_Upton) assembled a group of teachers, academics and computer enthusiasts to devise a computer to inspire children. The computer is inspired by Acorn's [BBC Micro](https://en.wikipedia.org/wiki/BBC_Micro) of 1981.The Model A, Model B and Model B+ names are references to the original models of the British educational Microcomputer, developed by [Acorn Computers](https://en.wikipedia.org/wiki/Acorn_Computers). The first ARM prototype version of the computer was mounted in a package the same size as a [USB memory stick](https://en.wikipedia.org/wiki/USB_memory_stick). It had a USB port on one end and an [HDMI](https://en.wikipedia.org/wiki/HDMI) port on the other.

**Raspberry Pi Technology**

The raspberry pi comes in two models, they are model A and model B. The main difference between model A and model B is USB port. Model A board will consume less power and that does not include an Ethernet port. But, the model B board includes an Ethernet port and designed in china. The raspberry pi comes with a set of open source technologies, i.e. communication and multimedia web technologies.In the year 2014, the foundation of the raspberry pi board launched the computer module, that packages a model B raspberry pi board into module for use as a part of embedded systems, to encourage their use.

## Raspberry Pi Hardware Specifications

The raspberry pi board comprises a program memory (RAM), processor and graphics chip, CPU, GPU, Ethernet port, GPIO pins, Xbee socket, UART, power source connector. And various interfaces for other external devices. It also requires mass storage, for that we use an SD flash memory card. So that raspberry pi board  will boot from this SD card similarly as a PC boots up into windows from its hard disk.

Essential hardware specifications of raspberry pi board mainly include SD card containing Linux OS, US keyboard, monitor, power supply and video cable. Optional hardware specifications  include USB mouse, powered USB hub, case, internet connection, the Model A or B: USB WiFi adaptor is used and internet connection to  Model B is LAN cable.

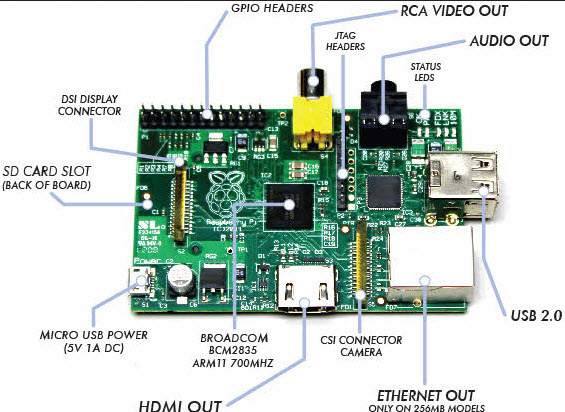


Fig. **Raspberry Pi**

#### Applications of Raspberry Pi

The raspberry pi boards are used in many applications like Media streamer, Arcade machine, Tablet computer, Home automation, Carputer, Internet radio, Controlling robots, Cosmic Computer, Hunting for meteorites, Coffee and also in raspberry pi based projects.

**Evolution of Beagle Board :**

1. **BeagleBone :**

Announced in the end of October 2011, the BeagleBone is a barebone development board with a Sitara ARM Cortex-A8 processor running at 720 MHz, 256 MB of RAM, two 46-pin expansion connectors, on-chip Ethernet, a microSD slot, and a USB host port and multipurpose device port which includes low-level serial control and JTAG hardware debug connections, so no JTAG emulator is required. The BeagleBone was initially priced at $89 USD.

2. **BeagleBone Black**

Launched in April 23, 2013 at a price of $45. Among other differences, it increases RAM to 512 MB, the processor clock to 1 GHz, and it adds HDMI and 2 GB of eMMC flash memory. The BeagleBone Black also ships with Linux kernal 3.8, upgraded from the original BeagleBone's Linux kernel 3.2, allowing the BeagleBone Black to take advantage of Direct Rendering Manager (DRM). BeagleBone Black Revision C (released in 2014) increased the size of the flash memory to 4 GB. This enables it to ship with Debian GNU/Linux installed. Previous revisions shipped with Ångström Linux.

3. **BeagleBoard-X15**

The BeagleBoard-X15 was planned for November 2015. It is based on the TI Sitara AM5728 processor with two ARM Cortex-A15 cores running at 1.5 GHz, two ARM Cortex-M4 cores running at 212 MHz and two TI C66x DSP cores running at 700 MHz. The used processor provides USB 3.0 support and has a PowerVR Dual Core SGX544 GPU running at 532 MHz.

**BeagleBoard Black**

The **BeagleBoard Black** is a low-power [open-source](https://en.wikipedia.org/wiki/Open-source_hardware) [single-board computer](https://en.wikipedia.org/wiki/Single-board_computer) produced by [Texas Instruments](https://en.wikipedia.org/wiki/Texas_Instruments) in association with [Digi-Key](https://en.wikipedia.org/wiki/Digi-Key) and [Newark element14](https://en.wikipedia.org/wiki/Newark_element14). The BeagleBoard was also designed with [open source software](https://en.wikipedia.org/wiki/Open_source_software) development in mind, and as a way of demonstrating the Texas Instrument's [OMAP3530](https://en.wikipedia.org/wiki/Texas_Instruments_OMAP) [system-on-a-chip](https://en.wikipedia.org/wiki/System-on-a-chip).[[8]](https://en.wikipedia.org/wiki/BeagleBoard" \l "cite_note-8) The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities. It is also sold to the public under the [Creative Commons](https://en.wikipedia.org/wiki/Creative_Commons) [share-alike](https://en.wikipedia.org/wiki/Share-alike) license. The board was designed using [Cadence](https://en.wikipedia.org/wiki/Cadence_Design_Systems) [OrCAD](https://en.wikipedia.org/wiki/OrCAD) for schematics and Cadence Allegro for PCB manufacturing; no simulation software was used.

### Features

A modified version of the BeagleBoard called the BeagleBoard-xM started shipping on August 27, 2010. The BeagleBoard-xM measures in at 82.55 by 82.55 mm and has a faster CPU core (clocked at 1 GHz compared to the 720 MHz of the BeagleBoard), more RAM (512 MB compared to 256 MB), onboard [Ethernet](https://en.wikipedia.org/wiki/Ethernet) jack, and 4 port USB hub. The BeagleBoard-xM lacks the onboard NAND and therefore requires the OS and other data to be stored on a microSD card. The addition of the Camera port to the -xM provides a simple way of importing video via Leopard Board cameras.

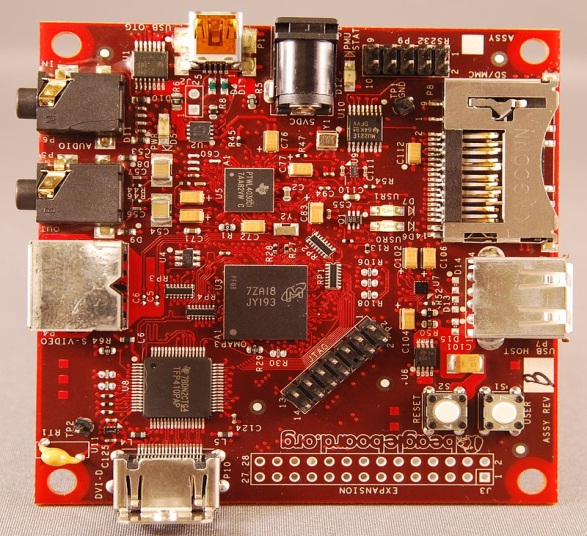


Fig.BeagleBoard Black

**Arduino**

Arduino is a collection of three things. There are Hardware prototype platform, Arduino language and IDE & libraries. [The Arduino boards are micro-controllers](https://www.elprocus.com/arduino-basics-and-design/), not a full-fledged computer. They don’t run a full operating system, but simply write the code and execute as their firmware interprets it.

**Evolution of Arduino :**

Arduino was created in Ivrea Italy as a Masters thesis project. The goal of Arduino was to allow nontechnical individuals to create technical projects of their own. The Arduino was intended to be affordable. Over 700,000 Arduino boards have been commercially produced since its founding.

1. **Serial Arduino**

• Release Year - 2005

• Processor - ATmega8

• Frequency - 16MHz

• Host Interface - DE-9 Serial Connection

• Uses RS232 as an interface for programming or communication with a computer.

• Specifically designed to be easily assembled with the most simple components.

2. **Arduino Nano**

• Release Date - May 15, 2008

• Processor - ATmega328 (ATmega168 before v3.0)

• Frequency - 16 MHz

• Host Interface - USB

• Uses a surface - mounted processor

• Lacks only a DC power jack and uses a Mini-B USB instead of standard on

3. **Arduino UNO**

**•** Release Date - September 24, 2010, • Processor - ATmega328P

• Frequency - 16 MHz, • Host Interface – USB, • Uses FTDI chip for USB

#### Advantages

Following are some of the main advantages of Arduino.

* Very easy to get started.
* Can be used for real-time applications for both hardware, software and IDE is open source.
* Not much programming knowledge needed to do basic stuff.
* It is very easy to extend and has tons of user contributed shields and libraries. Shields are available to do attractive much anything.

**Difference between Arduino, Raspberry Pi and BeagleBone Black :**

The Arduino, Raspberry Pi, BeagleBone and PCDuino may look quite similar for you, but they are in fact very different devices. The Arduino is a microcontroller. A microcontroller is just one tiny part of a computer. The arduino can be programmed in C, but can’t run an operating system.



**Conclusion**:

Thus we have understood the concept of Raspberry-Pi, BeagleBoard Black, Arduino and Microcontroller.

**Assignment** **No.** A**2**

**Problem Statement:** Study of different operating systems for Raspberry-Pi /Beagle board. Understanding the process of OS installation on Raspberry-Pi /Beagle board

**Software/Hardware** **Requirements:**

• Windows 10

• Ubuntu 16.04

•Beagle Bone Black kit

**Objectives:**

To learn Basic concepts of Beagle Bone and Raspberry pi the process of OS installation on Beagle board

**Theory:**

The BeagleBone Black includes a 2GB or 4GB on-board eMMC flash memory chip. It comes with the Debian distribution factory pre-installed. You can flash new operating systems including Angstrom, Ubuntu, Android, and others. The following pages will illustrate the steps to getting the latest of each type of supported distribution onto the on-board eMMC. In addition to the eMMC, you can also boot directly from a microSD card similarly to the original BeagleBone.

**Operating Systems**

The Raspberry Pi Foundation recommends the use of [Raspbian](https://en.wikipedia.org/wiki/Raspbian), a [Debian](https://en.wikipedia.org/wiki/Debian)-based [Linux](https://en.wikipedia.org/wiki/Linux) operating system. Other third-party operating systems available via the official website include [Ubuntu MATE](https://en.wikipedia.org/wiki/Ubuntu_MATE), [Windows 10 IoT Core](https://en.wikipedia.org/wiki/Windows_10_IoT_Core), [RISC OS](https://en.wikipedia.org/wiki/RISC_OS) and specialized distributions for the [Kodi](https://en.wikipedia.org/wiki/Kodi_(software)) media centre and classroom management. Many other operating systems can also run on the Raspberry Pi.

**Other operating systems (not Linux-based)**

* [RISC OS](https://en.wikipedia.org/wiki/RISC_OS) Pi (a special cut down version RISC OS Pico, for 16 MB cards and larger for all models of Pi 1 & 2, has also been made available.)
* [FreeBSD](https://en.wikipedia.org/wiki/FreeBSD)
* [NetBSD](https://en.wikipedia.org/wiki/NetBSD)
* [Plan 9 from Bell Labs](https://en.wikipedia.org/wiki/Plan_9_from_Bell_Labs) and [Inferno](https://en.wikipedia.org/wiki/Inferno_(operating_system)) (in beta)
* [Windows 10 IoT Core](https://en.wikipedia.org/wiki/Windows_10_IoT_Core) – a no-cost edition of [Windows 10](https://en.wikipedia.org/wiki/Windows_10) offered by Microsoft that runs natively on the Raspberry Pi 2.
* [xv6](https://en.wikipedia.org/wiki/Xv6)– is a modern reimplementation of Sixth Edition Unix OS for teaching purposes; it is ported to Raspberry Pi from MIT xv6; this xv6 port can boot from [NOOBS](https://en.wikipedia.org/w/index.php?title=NOOBS&action=edit&redlink=1).
* [Haiku](https://en.wikipedia.org/wiki/Haiku_(operating_system)) – is an opensource BeOS clone that has been compiled for the Raspberry Pi and several other ARM boards. Work on Pi 1 began in 2011, but only the Pi 2 will be supported.
* [HelenOS](https://en.wikipedia.org/wiki/HelenOS) – a portable microkernel-based multiserver operating system; has basic Raspberry Pi support since version 0.6.0

**Other operating systems (Linux-based)**

* [Android Things](https://en.wikipedia.org/wiki/Android_Things) – an embedded version of the [Android](https://en.wikipedia.org/wiki/Android_(operating_system)) operating system designed for [IoT](https://en.wikipedia.org/wiki/Internet_of_things) device development.
* [Arch Linux ARM](https://en.wikipedia.org/wiki/Arch_Linux_ARM) – a port of [Arch Linux](https://en.wikipedia.org/wiki/Arch_Linux) for [ARM](https://en.wikipedia.org/wiki/ARM_architecture) processors.
* [openSUSE](https://en.wikipedia.org/wiki/OpenSUSE)
* [SUSE Linux Enterprise](https://en.wikipedia.org/wiki/SUSE_Linux_Enterprise) Server 12 SP2
* Raspberry Pi [Fedora](https://en.wikipedia.org/wiki/Fedora_(operating_system)) Remix
* [Gentoo Linux](https://en.wikipedia.org/wiki/Gentoo_Linux)
* [CentOS](https://en.wikipedia.org/wiki/CentOS) for Raspberry Pi 2 and later
* [Devuan](http://devuan.org) - a version of Debian with [sysvinit](https://en.wikipedia.org/wiki/Sysvinit) instead of [systemd](https://en.wikipedia.org/wiki/Systemd)
* [RedSleeve](https://en.wikipedia.org/wiki/RedSleeve) (a RHEL port) for Raspberry Pi 1
* [Slackware ARM](https://en.wikipedia.org/wiki/Slackware_ARM) – version 13.37 and later runs on the Raspberry Pi without modification. The 128–496 MB of available memory on the Raspberry Pi is at least twice the minimum requirement of 64 MB needed to run Slackware Linux on an ARM or i386 system. (Whereas the majority of Linux systems boot into a [graphical user interface](https://en.wikipedia.org/wiki/Graphical_user_interface), Slackware's default user environment is the [textual shell](https://en.wikipedia.org/wiki/Shell_(computing)) / [command line interface](https://en.wikipedia.org/wiki/Command_line_interface).) The [Fluxbox](https://en.wikipedia.org/wiki/Fluxbox) window manager running under the [X Window System](https://en.wikipedia.org/wiki/X_Window_System) requires an additional 48 MB of RAM.
* [OpenWrt](https://en.wikipedia.org/wiki/OpenWrt) – is primarily used on embedded devices to route network traffic.
* [Kali Linux](https://en.wikipedia.org/wiki/Kali_Linux) – is a Debian-derived distro designed for digital forensics and penetration testing.
* [SolydXK](https://en.wikipedia.org/wiki/SolydXK) – is a light Debian-derived distro with Xfce.
* [Ark OS](https://en.wikipedia.org/wiki/Ark_OS) – is designed for website and email self-hosting.
* [Sailfish OS](https://en.wikipedia.org/wiki/Sailfish_OS) with Raspberry Pi 2 (due to use ARM Cortex-A7 CPU; Raspberry Pi 1 uses different ARMv6 architecture and Sailfish requires ARMv7.)
* [Tiny Core Linux](https://en.wikipedia.org/wiki/Tiny_Core_Linux) – a minimal [Linux](https://en.wikipedia.org/wiki/Linux) operating system focused on providing a base system using [BusyBox](https://en.wikipedia.org/wiki/BusyBox) and [FLTK](https://en.wikipedia.org/wiki/FLTK). Designed to run primarily in [RAM](https://en.wikipedia.org/wiki/RAM).
* [Alpine Linux](https://en.wikipedia.org/wiki/Alpine_Linux) – is a [Linux distribution](https://en.wikipedia.org/wiki/Linux_distribution) based on [musl](https://en.wikipedia.org/wiki/Musl) and [BusyBox](https://en.wikipedia.org/wiki/BusyBox), primarily designed for "[power users](https://en.wikipedia.org/wiki/Power_users) who appreciate security, simplicity and resource efficiency".
* [Void Linux](https://en.wikipedia.org/wiki/Void_Linux) – a [rolling release](https://en.wikipedia.org/wiki/Rolling_release) Linux distribution which was designed and implemented from scratch, provides images based on musl or [glibc](https://en.wikipedia.org/wiki/Glibc).
* [Fedora](https://en.wikipedia.org/wiki/Fedora_(operating_system)) 25 – supports Pi 2 and later (Pi 1 is supported by some unofficial derivatives, e.g. listed here.).
* Media centre operating systems
* [Daylight Linux](https://en.wikipedia.org/wiki/Daylight_Linux) – An ultra-lightweight operating system with the Fluxbox interface
* [Raspberry Digital Signage](https://en.wikipedia.org/w/index.php?title=Raspberry_Digital_Signage&action=edit&redlink=1) – An operating system designed for digital signage deployments.

**OS Installation :**

**Step** **1:** **Download** **the** **latest** **software** **image**

Download the lastest Debian image from beagleboard.org/latest-images. The "IoT" images provide more free disk space if you don't need to use a graphical user interface (GUI). The Debian distribution is provied for the boards. The file you download will have an .img.xz extension. This is a compressed sector-by-sector image of the SD card.

**Step** **2:** **Install** **SD** **card** **programming** **utility** **Download** **and** **install** **Etcher**.

Some general help on programming SD cards can be found on the Ubuntu Image Writer page.

**Step** **3:** **Connect** **SD** **card** **to** **your** **computer**

Use your computer's SD slot or a USB adapter to connect the SD card to your computer.

**Step** **4:** **Write** **the** **image** **to** **your** **SD** **card**

Use Etcher to write the image to your SD card. Etcher will transparently decompress the image on-the-fly before writing it to the SD card.

**Step** **5:** **Eject** **the** **SD** **card**

Eject the newly programmed SD card.

**Step** **6:** **Boot** **your** **board** **off** **of** **the** **SD** **card**

Insert SD card into your (powered- down) board, hold down the USER/BOOT button (if using Black) and apply power, either by the USB cable or 5V adapter.

**Conclusion:**

We have studied different operating systems for Raspberry-Pi /Beagle board and the process of OS installation on Beagle board

**Assignment Number: - A-3**

**Problem Statement:** Study of Connectivity and configuration of Raspberry-Pi /Beagle board circuit with basic Peripherals, LEDS. Understanding GPIO and its use in program.

**Software/Hardware Requirements**

• Windows 10, 8.1 Update, 8, or 7.

• Ubuntu 16.04

• Beagle Bone Black kit

**Objectives:**

To learn Connectivity and configuration of Raspberry-Pi /Beagle board circuit with basic peripherals, LEDS ,GPIO and its use in program.

**Theory:**

**General-purpose input/output** (**GPIO**) is a generic pin on an integrated circuit or computer board whose behaviour—including whether it is an input or output pin—is controllable by the user at run time.

**How to connect your beaglebone via ssh over usb**

a. In Mac OS X and Linux, you can use the default terminal window to establish an SSH

communication, because SSH comes in all Unix-based OSes. Follow these steps:Connect your BeagleBone to your computer by using a Mini USB cable.

b. Open a new window, as follows:

c. On a Mac, navigate to /Applications/Utilities and double-click Terminal to open a new terminal window.

d. In Linux, press Ctrl+Alt+T to open a new terminal window.

e. Type sudo ssh root@192.168.7.2.

f. Enter your computer password, and type yes.

g. When you’re asked to type a password, press Enter or Return.

**The color-coding is as follows:**

a. Pins in green are pre-assigned by the default device tree configuration and the hw io BeagleBone Black driver, so you can use these directly (applies to I2C2 which is the "i2c" module in hwio, and the analog pins.)

b. Pins in blue are not pre-assigned, and you can make them GPIO pins by using PinMode.

c. Pins in red are pre-assigned by the default device tree configuration on the BeagleBone Black. These are not available to use as GPIO unless you use a custom device tree.

d. Pins can be referred to by their number, e.g. P8.37 (port 8, pin 37), or by their function, e.g. GPIO2\_14. All GPIO pins that can be assigned support digital input and output. AIN pins (AIN0-AIN6) are analog input only pins.

**Important**

All GPIO pins are rated 0 - 3.3V only, and can draw 4-6mA.

All AIN pins are rated 0 - 1.8V only.

Exceeding these voltages is likely to damage the device.

Pins Description for power supply or voltage reference pins are as follows:

GND - Ground, the 0V reference

VDD 3.3V - 3.3V with maximum current 250mA

VDD 5V - 5V with maximum current 1000mA, but only if the BeagleBone is powered by an external power source, and only to the supply current limit. SYS 5V - 5V with maximum current 250mA

VDD ADC(1.8V) - a 1.8V reference voltage for circuits supplying analog values to AIN pins.

This is only a reference voltage; it should not be used as a current source.

GNDA\_ADC - 0V reference for analog/digital conversion.

**Blinking an led with beagle bone black**

In this practical you will learn how to control an external LED using a BeagleBone Black (BBB) from Python. Because the BBB runs Linux, there are many ways in which it can be programmed. In this practical we show how to control an output pin and hence LED using Python.

**Wiring**

Steps for wiring are given below:

1. Push the LED leads into the breadboard, with the longer (positive) lead towards the top of the breadboard. It does not matter which way around the resistor goes.

2. The top two connections on the BBB expansion header we are using (P8) are both GND. The other lead is connected to pin 10, which is the right- hand connector on the fifth row down. The pins are numbered left to right, 1, 2 then on the next row down 3,4 etc.

**Program**

**Command to create new file**

# nano blink.py

**code**

1. import Adafruit\_BBIO.GPIO as GPIO
2. import time
4. GPIO.setup("P8\_10", GPIO.OUT)
6. while True:
7. GPIO.output("P8\_10", GPIO.HIGH)
8. time.sleep(0.5)
9. GPIO.output("P8\_10", GPIO.LOW)
10. time.sleep(0.5)

Save and exit the editor using CTRL-x and the Y to confirm.

Run command

# python blink.py

**Conclusion:**

We have studied Connectivity and configuration of Raspberry-Pi /Beagle board circuit with basic peripherals, LEDS and GPIO and its use in program

Assignment Number: - B1

**Problem Statement:**

Understanding connectivity of Beagle bone board circuit with IR sensor. Write an application to detect obstacle and notify user using LEDs.

**Theory:**

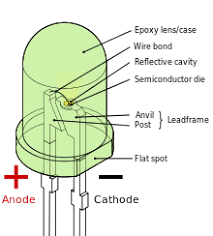
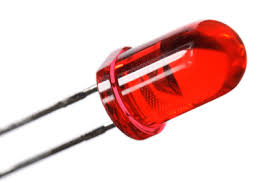
Infrared radiation is an electromagnetic wave with wavelength of 700nm to 1 mm. It is emitted by objects with temperature above 0 kelvins. Furthermore intensity and wavelength of infrared radiation depends on the temperature of the object.

The infrared sensors are the sensors that detect/measure infrared radiation or change in the radiation from outer source or inbuilt source. Also sensors that use the property of infrared radiations to detect the changes in surrounding are termed as infrared sensors.

An infrared sensor is an electronic instrument which is used to sense certain characteristics of its surroundings by either emitting and/or detecting infrared radiation. Infrared sensors are also capable of measuring the heat being emitted by an object and detecting motion.

A **light-emitting diode** (**LED**) is a two-lead semiconductor light source. It is a p–n junction diode that emits light when activated When a suitable current is applied to the leads; electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence, and the color of the light (corresponding to the energy of the photon) is determined by the energy band gap of the semiconductor. LEDs are typically small (less than 1 mm2) and integrated optical components may be used to shape the radiation pattern.

LEDs have many advantages over incandescent light sources, including lower energy consumption, longer lifetime, improved physical robustness, smaller size, and faster switching. Light-emitting diodes are used in applications as diverse as aviation lighting, automotive headlamps, advertising, general lighting, traffic signals, camera flashes, lighted wallpaper and medical devices. They are also significantly more energy efficient and, arguably, have fewer environmental concerns linked to their disposal

A **buzzer** or **beeper** is an audio signalling device, which may be mechanical, electromechanical, or piezoelectric (*piezo* for short). Typical uses of buzzers and beepers include alarm devices, timers, and confirmation of user input such as a mouse click or keystroke. There are three types of buzzers: Electromechanical, Mechanical and Piezoelectric. Modern applications of buzzers include Novelty uses, judging panels, educational purposes, etc.

When an obstacle is detected by the IR sensors it returns a zero value. The led glows and buzzer buzzes. When there is no obstacle the IR sensors keeps on returning a non zero value. Thus the LED does not turn on and the buzzer is also off.

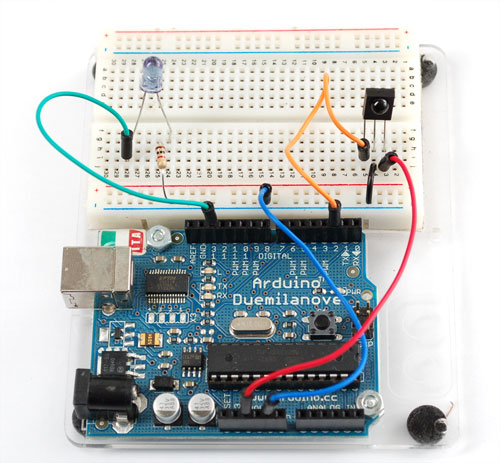


**Connections:**

Connect the LED, IR sensor, resistor and buzzer on the bread board.

For IR sensor, connect the GND pin to P9\_1 of beagle-bone, Vcc to P9\_5 and Data input to P9\_12.

The Vcc pin of LED is connected to P8\_7 and GND to P9\_2.

The Vcc and GND pins of the buzzer are connected to P8\_8 and P8\_1 respectively.

**Algorithm:**

1. import Adafruit\_BBIO.GPIO as GPIO
2. import time
3. sensor 🡨 P9\_12, led 🡨 P8\_7, buz 🡨 P8\_8
4. Set the pin that is connected to IR sensor in INPUT mode. (P9\_12)
5. Set the pins that are connected to the led and buzzer in OUTPUT mode.(P8\_7, P8\_8)
6. if (GPIO.input(sensor)==0):
   1. GPIO.output(led,GPIO.HIGH) i.e.switch on the led
   2. GPIO.output(buz,GPIO.HIGH) i.e.switch on the buzzer
   3. time.sleep(0.5)
7. else:
   1. GPIO.output(led,GPIO.LOW) i.e.switch off the led
   2. GPIO.output(led,GPIO.LOW) i.e.switch off the buzzer

**Input:**

Any obstacle that comes in the range of the sensor is detected. IR sensors actually measure the heat being emitted from the object. So the heat is the actual input for the sensors.

The IR sensor gets its input from pin number P9\_12.

**Output:**

The output is shown by the LED and the buzzer. When an obstacle is detected, the LED glows and buzzer is turned on, i.e. whenever heat is sensed by the sensor, output is shown.

The output is shown by making the pins P8\_7 (led) and P8\_8 high (buzzer).

**Conclusion:**

We have successfully implemented the connection of IR sensor and Beagle bone for obstacle detection. The output is shown by a glowing LED and buzzer.

**Assignment Number :- B-2**

**Problem Statement**

Understanding and Connectivity of Raspberry-Pi/Beagle board with camera. Write an application to capture and store the image.

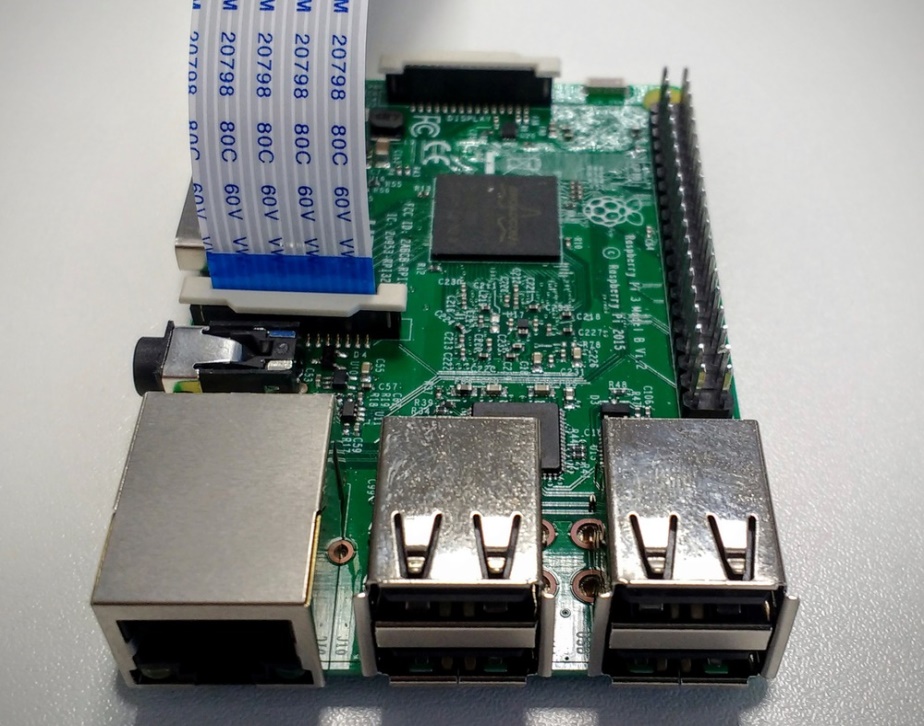
**Theory**

The Raspberry Pi camera module is capable of taking full HD 1080p photo and video and can be controlled programmatically. The new 8-megapixel Pi camera comes in two varieties: a standard visible light camera and an infrared-sensitive camera known as the Pi NoIR. The Pi camera is capable of taking still pictures up to a resolution of 3280 x 2464 pixels, and video up to 1080p at 30fps.

**Connecting the Camera**

The camera board attaches to the Raspberry Pi using a CSI flat ribbon cable, which comes with the camera when purchased. It can then be attached to the CSI port on the Raspberry Pi, as shown below.

The flex cable inserts into the connector situated between the Ethernet and HDMI ports, with the silver connectors facing the HDMI port. The flex cable connector should be opened by pulling the tabs on the top of the connector upwards then towards the Ethernet port. The flex cable should be inserted firmly into the connector, with care taken not to bend the flex at too acute an angle. The top part of the connector should then be pushed towards the HDMI connector and down, while the flex cable is held in place.



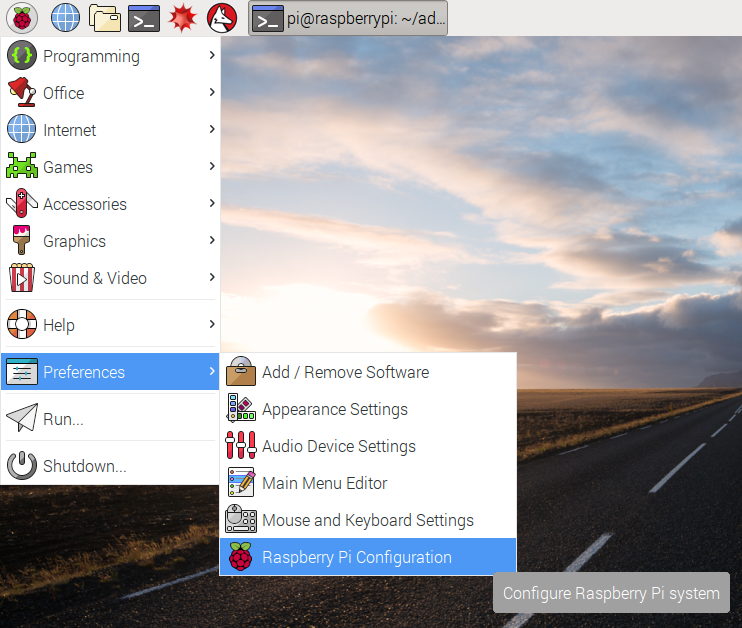
**Enabling the Camera**

1) Open the **raspi-config** tool from the Terminal

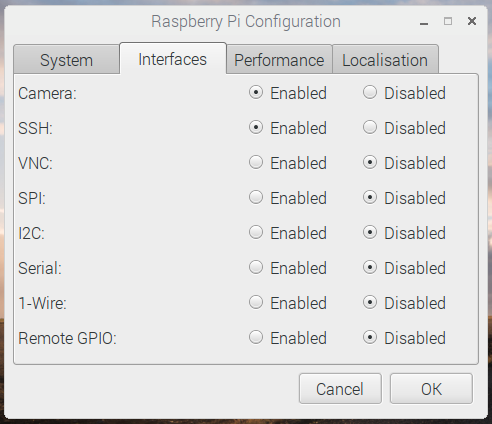
**sudo raspi-config**

2) Select **Enable camera** and hit **Enter**, then go to **Finish** and you'll be prompted to reboot.

You can also enable the Pi camera, using the configuration menu. Open this by clicking on **Menu** > **Preferences** > **Raspberry Pi Configuration**:



Then click on the radio button to enable the camera:



**Capture Images**

You can then activate the camera from the terminal to take photos with the following command:

**raspistill -o cam.jpg**

To take videos, use this command:

**raspivid -o vid.h264**

If you want greater control over the photos and videos you take, you can use Python 3 to control the camera. This can be done with the help PiCamera package in Python.

**PiCamera**

This package provides a pure Python interface to the Raspberry Pi camera module for Python 2.7 (or above) or Python 3.2 (or above).

To install picamera on Raspbian, it is best to use the system’s package manager: apt. This will ensure that picamera is easy to keep up to date, and easy to remove should you wish to do so. It will also make picamera available for all users on the system. To install picamera using apt simply run:

**$ sudo apt-get update**

**$ sudo apt-get install python-picamera python3-picamera**

To upgrade your installation when new releases are made you can simply use apt’s normal upgrade procedure:

**$ sudo apt-get update**

**$ sudo apt-get upgrade**

If you ever need to remove your installation:

**$ sudo apt-get remove python-picamera python3-picamera**

**To Capture images with PiCamera**

First, at the Python prompt or at the top of a Python script, enter:

**import picamera**

This will make the library available to the script. Now create an instance of the PiCamera class:

**camera = picamera.PiCamera()**

And take a picture:

**camera.capture('image.jpg')**

### Horizontal and Vertical flip

Like with the raspistill command, you can apply a horizontal and vertical flip if your camera is positioned upside-down. This is done by changing the hflip and vflip properties directly:

**camera.hflip = True**

**camera.vflip = True**

Be sure to use an upper case T in True as this is a keyword in Python.

### Preview

You can display a preview showing the camera feed on screen. Warning: this will overlay your Python session by default; if you have trouble stopping the preview, simply pressing Ctrl+D to terminate the Python session is usually enough to restore the display:

**camera.start\_preview()**

You can use the stop\_preview method to remove the preview overlay and restore the display:

**camera.stop\_preview()**

Alternatively, you can access the Pi using [SSH](https://www.raspberrypi.org/documentation/remote-access/ssh/README.md) from another computer, open a Python prompt and enter these commands, displaying the preview on the monitor connected to the Pi (not the computer you're connected from).

### Camera settings

You can change other camera configuration by editing property values, for example:

**camera.brightness = 70**

This will change the brightness setting from its default 50 to 70 (values between 0 and 100).

Other settings are available. Here is a list with their default values:

camera.sharpness = 0

camera.contrast = 0

camera.brightness = 50

camera.saturation = 0

camera.ISO = 0

camera.video\_stabilization = False

camera.exposure\_compensation = 0

camera.exposure\_mode = 'auto'

camera.meter\_mode = 'average'

camera.awb\_mode = 'auto'

camera.image\_effect = 'none'

camera.color\_effects = None

camera.rotation = 0

camera.hflip = False

camera.vflip = False

camera.crop = (0.0, 0.0, 1.0, 1.0)

### Sleep

### You can add pauses between commands using sleep from the time module:

import picamera

from time import sleep

camera = picamera.PiCamera()

camera.capture('image1.jpg')

sleep(5)

camera.capture('image2.jpg')

### Video recording

Record 5 seconds of video:

camera.start\_recording('video.h264')

sleep(5)

camera.stop\_recording()

**Algorithm**

1. Import the RPi.GPIO, PiCamera and sleep libraries.
2. Set the GPIO pins 17(input mode) and 27(output mode).
3. Check if the input is “0”.
   1. Write “1” to GPIO pin 27 as output(LED is on).

gpio.output(27,1)

* 1. Create PiCamera object camera.

camera = PiCamera()

* 1. Start preview.

camera.start\_preview()

* 1. Sleep for a minimum of 2 seconds
  2. Capture the image.

camera.capture('image.jpg', use\_video\_port=True)

* 1. Stop the preview and Close Camera

camera.stop\_preview()

camera.close()

* 1. Write “0” to GPIO pin 27 as output(LED is off).

gpio.output(27,0)

**NOTE :** Ensure that the Video Port is set to TRUE for Camera module to work properly.

**Input and Output**

The LED glows when the preview is started. The Camera captures and stores the image at the path mentioned in the capture function. Once the image is captured LED is turned off.

**Conclusion**

Thus we have successfully understood the connectivity of Raspberry-Pi with Camera Module to capture images.

**CODE**

import RPi.GPIO as gpio

from time import sleep

from picamera import PiCamera

gpio.setmode(gpio.BCM)

gpio.setup(17,gpio.IN)

gpio.setup(27,gpio.OUT)

if(gpio.input(17)==0):

gpio.output(27,1)

camera = PiCamera()

camera.start\_preview()

camera.vflip = True

sleep(2)

camera.capture('foo.jpg', use\_video\_port=True)

camera.stop\_preview()

camera.close()

gpio.output(27, 0)

Assignment: B3

**Title:** To write an application to and demonstrate the change in BeagleBoard/ ARM Cortex A5 Microprocessor /CPU frequency or square wave of pro-grammable frequency.

**Objective**

To understand the basic working principle of DC motors.

To understand how to write programs for Beaglebone Black in Python.

**Theory**

**DC motors**

A DC motor in simple words is a device that converts direct current(electrical energy) into mechanical energy.Its movement is produced by the physical behavior of electromagnetism.DC motors have only two terminals. If applied a voltage to these terminals the motor will run, if invert the terminals position the motor will change its direction. If the motor is running and you suddenly disconnectboth terminals themotorwillkeeprotatingbut slowing down until stopping. Finally if the motor is running and if suddenly short-circuit both terminals the motor will stop.The most common types rely on the forces produced by magnetic elds.

Types of DC motors:

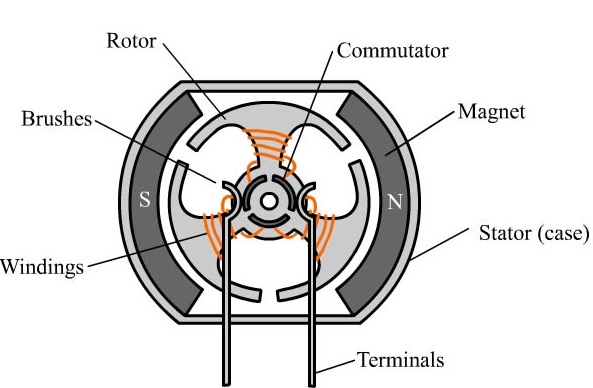


Figure 1: DC Motor

1. Brushed:

The brushed DC electric motor generates torque directly from DC power supplied to the motor by using internal commutation, stationary magnets (permanent or electromagnets), and rotating electrical mag-nets.

2. Brushless:

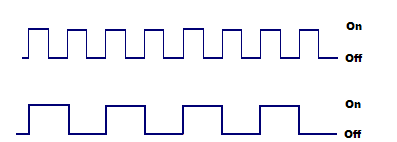
Typical brushless DC motors use a rotating permanent magnet in the rotor, and stationary electrical current/coil magnets on the motor hous-ing for the stator.

Pulse Width Modulation

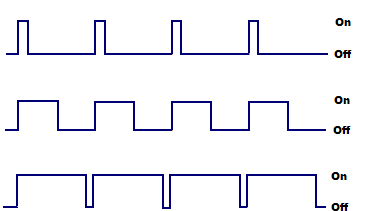
A DC motor’s speed can be controlled over a wide range, using either a variable supply voltage or by changing the strength of current in its eld windings.PWM(Pulse width modulation) is one such method whos main aim is to allow the control of the power supplied to electrical devices by varying the duty cycle.

The main advantage of PWM is that power loss in the switching devices is very low. When a switch is o there is practically no current, and when it is on and power is being transferred to the load, there is almost no voltage drop across the switch. Power loss, being the product of voltage and current, is

thus in both cases close to zero. PWM also works well with digital controls, which, because of their on/o nature, can easily set the needed duty cycle. In this method, we can either vary:



1. The Frequency

Figure 2: PWM: same duty cycle, dierent frequencies.

2. The Duty cycle

Figure 3: PWM: same frequency, different duty cycles

Duty Cycles

The term duty cycle describes the proportion of ’on’ time to the regular inter-val or ’period’ of time; a low duty cycle corresponds to low power, because the power is o for most of the time. Duty cycle is expressed in percent, 100% being fully on.

Most radio frequency (RF) measurements are either continuous wave (CW) or pulsed RF. CW RF is uninterrupted RF such as from an oscil-lator. Amplitude modulated (AM), frequency modulated (FM), and phase modulated (PM) RF are considered CW since the RF is continuously present. The power may vary with time due to modulation, but RF is always present. Pulsed RF, on the other hand, is bursts (pulses) of RF with no RF present between bursts. The most general case of pulsed RF consists of pulses of a xed pulse width (PW) which come at a xed time interval, or period, (T).Pulses at a xed interval of time arrive at a rate or frequency referred to as the pulse repetition frequency (PRF) of so many pulse per second. Pulse repetition interval (PRI) and PRF are reciprocals of each other.

PRF = 1=T = 1=PRI................................................................................[1]

Power measurements are classied as either peak pulse power, Pp, or av-erage power, Pave. The average value is dened as that level where the pulse area above the average is equal to area below average between pulses.The area of the pulse is the pulse width multiplied by the peak pulse power. The average area is equal to the average value of power multiplied by the pulse period.

Since the two values are equal:

Pave xT=Pp xPW......................................................................................[2] or

Pave/Pp =PW/T.........................................................................................[3] Using [1]

Pave/Pp =PW/T=PWxPRF=PW/PRI=dutycycle.............................[4]

The ratio of the average power to the peak pulse power is the duty cycle and represents the percentage of time the power is present.

In the case of a square wave the duty cycle is 0.5 (50%) since the pulses are present 1/2 the time, the denition of a square wave. Form g, the pulse width is 1 unit of time and the period is 10 units. In this case the duty cycle is: PW/T = 1/10 = 0.1 (10%).

Besides expressing duty cycle as a ratio as obtained in equation [4], it is commonly expressed as either a percentage or in decibels (dB). To express the duty cycle of equation [4] as a percentage, multiply the value obtained by 100 and add the percent symbol. Thus a duty cycle of 0.001 is also 0.1%.

The duty cycle can be expressed logarithmically (dB) so it can be added to or subtracted from power measured in dBm/dBW rather than converting to, and using absolute units.

Duty cycle (dB) = 10 log(duty cycle ratio)............................................[5]

For the example of the 0.001 duty cycle, this would be 10 log(0.001) = -30 dB.

Thus the average power would be 30 dB less than the peak power.

Python Programming Adafruit BBIO.PWM Library

Setup

To setup a pin to use PWM:

import Adafruit\_BBIO.PWM as PWM

PWM.start(channel, duty, freq=2000, polarity=0)

PWM.start("P9\_14", 50)

#optionally, you can set the frequency as well as #the polarity from their defaults:

PWM.start("P9\_14", 50, 1000, 1)

Duty cycle or the frequency can be set as follows:

PWM.set\_duty\_cycle("P9\_14", 25.5)

PWM.set\_frequency("P9\_14", 10)

To either disable that specic channel, or cleanup:

PWM.stop("P9\_14")

PWM.cleanup()

Pinout Diagram

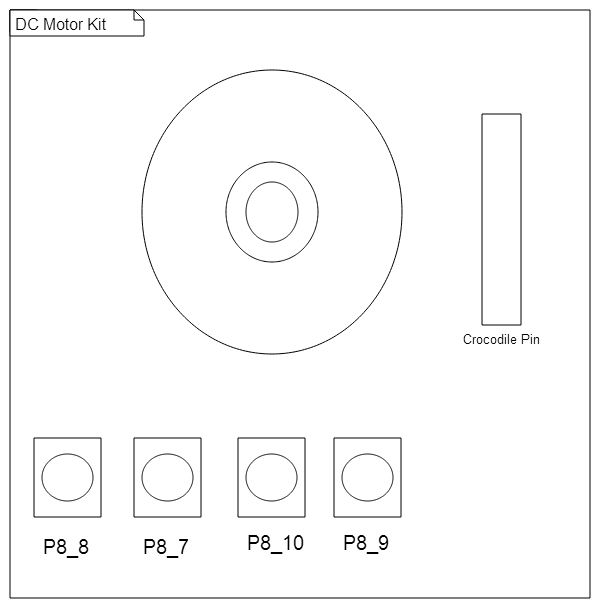


Figure 4: Pinout Diagram

Algorithm (Duty Cycle)

1. Import appropriate libraries into the program.

import Adafruit\_BBIO.GPIO as GPIO import Adafruit\_BBIO.PWM as PWM import time

2. Set all the pins on the kit to IN mode.

GPIO.setup("P8\_7",GPIO.IN)

3. Start PWM.

PWM.start("P8\_13")

4. Loop begins

(a) If input at pin P8 ~~8~~ is 0, then set Duty cycle at pin P8 13 as 20. (b) If input at pin P8 ~~7~~ is 0, then set Duty cycle at pin P8 13 as 90. (c) If input at pin P8 10 is 0, then set Duty cycle at pin P8 13 as 50. (d) If input at pin P8 ~~9~~ is 0, then set Duty cycle at pin P8 13 as 0.

Example:

if (GPIO.input("P8\_8")==0):

PWM.set\_duty\_cycle("P8\_13", 20)

elif (GPIO.input("P8\_7")==0):

PWM.set\_duty\_cycle("P8\_13", 90)

elif (GPIO.input("P8\_10")==0):

PWM.set\_duty\_cycle("P8\_13", 50)

elif (GPIO.input("P8\_9")==0):

PWM.set\_duty\_cycle("P8\_13",0)

5. Loop Ends

Algorithm (Frequency)

1. Import appropriate libraries into the program.

import Adafruit\_BBIO.GPIO as GPIO import Adafruit\_BBIO.PWM as PWM import time

2. Set all the pins on the kit to IN mode.

GPIO.setup("P8\_7",GPIO.IN)

3. Start PWM.

PWM.start("P8\_13",50,50)

4. Loop begins

(a) If input at pin P8 ~~8~~ is 0, then set Frequency at pin P8 13 as 5. (b) If input at pin P8 ~~7~~ is 0, then set Frequency at pin P8 13 as 50. (c) If input at pin P8 10 is 0, then set Frequency at pin P8 13 as 100. (d) If input at pin P8 ~~9~~ is 0, then set Frequency at pin P8 13 as 0.

Example:

if (GPIO.input("P8\_8")==0): PWM.set\_frequency("P8\_13", 5)

elif (GPIO.input("P8\_7")==0): PWM.set\_frequency("P8\_13", 50) elif (GPIO.input("P8\_10")==0): PWM.set\_frequency("P8\_13", 100) elif (GPIO.input("P8\_9")==0): PWM.set\_frequency("P8\_13", 1)

5. Loop Ends

**Input and Output**

|  |  |  |
| --- | --- | --- |
| Duty Cycle | Frequency | Result |
| 0 | 2 | No Rotation |
| 1 | 2 | Slow |
| 20 | 2 | Speed Increases |
| 60 | 2 | Speed Increases |
| 100 | 2 | Top Speed |
| 60 | 1 | Maximum Fluctuation |
| 60 | 20 | Low Fluctuation |
| 60 | 100 | Very Low Fluctuation |

**Conclusion**

Thus, studied DC Motor program and its practical implementation using Beaglebone Black Board in python.

Assignment Number: C-1

**Problem Statement**

Write an application using Raspberry-Pi /Beagle board to control the operation of stepper motor.

**Objective**

* To study working principle of stepper motor.
* To understand how to write stepper motor program for Beagle bone Black in Python.

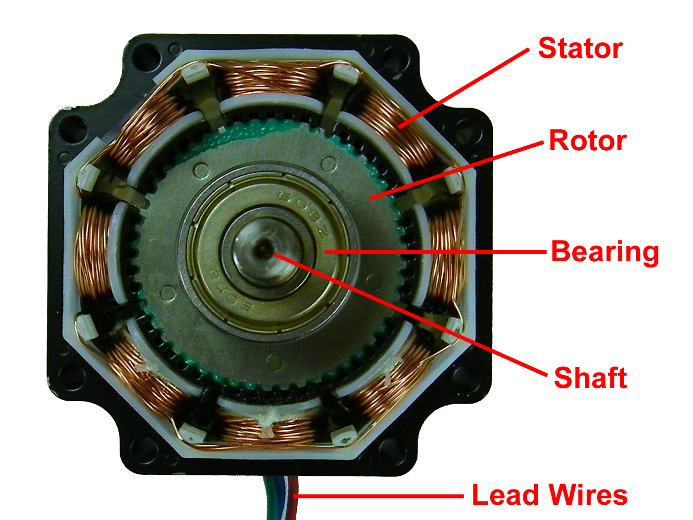
**Theory**

**Stepper motor:**

A stepper motor has two parts. A rotor (part that rotates) and a stat (stationary).In general, the stator holds the windings/coils (or electromagnets) and the rotor is a permanent magnet attached to the shaft of the motor.

Working of Stepper Motor

The main use of stepper motors is to control motion, whether it is linear or rotational. In the case of rotational motion, receiving digital pulses in a correct sequence allows the shaft of a stepper motor to rotate in discrete step increments. A pulse (also referred to as a clock or step signal) used in a stepper motor system can be produced by microprocessors, timing logic, a toggle switch or relay closure. A train of digital pulses translates into shaft revolutions. Each revolution requires a given number of pulses and each pulse equals one rotary increment or step, which is only a portion of one complete rotation. There are numerous relationships between the motors shaft rota-tion and input pulses. One such relationship is the direction of rotation and the sequence of applied pulses. With proper sequential pulses being deliv-ered to the device, the rotation of the shaft motor will undergo a clockwise or counterclockwise rotation. Another relation between the motor’s rotation and input pulses is the relationship between frequency and speed. Increasing the frequency of the input pulses allows for the speed of the motor shaft rotation to increase.



Some important formulae:

1. Step angle calculation:

’ = ((Ns Nr)=(Ns Nr)) 360 Where:-

* = Step Angle.

Ns = Number of teeth on stator.

Nr = Number of teeth on rotor.

1. Steps per second:

Step per second = (rpm steps per revolution) 60 Where:-

rpm = Revolutions per minute.

**Excitation Modes**

Commonly used excitation modes for step motors are as below:

1. Full step operation, the motor moves through its basic step angle, i.e., a 13.84 step motor takes 200 steps per motor revolution. There are two types of full step excitation modes.

In single phase mode, also known as one-phase on, full step excitation, the motor is operated with only one phase (group of windings) energized at a time. This mode requires the least amount of power from the driver of any of the excitation modes.

In dual phase mode, also known as two-phase on, full step excitation, the motor is operated with both phases energized at the same time. This mode provides improved torque and speed performance. Dual phase excitation provides about 30% to 40% more torque than single phase excitation, but does require twice as much power from the driver.

1. Half step excitation is alternating single and dual phase operation resulting in steps that are half the basic step angle. Due to the smaller step angle, this mode provides twice the resolution and smoother operation. Half stepping produces roughly 15% less torque than dual phase full stepping. Modified half stepping eliminates this torque decrease by increasing the current applied to the motor when a single phase is energized.

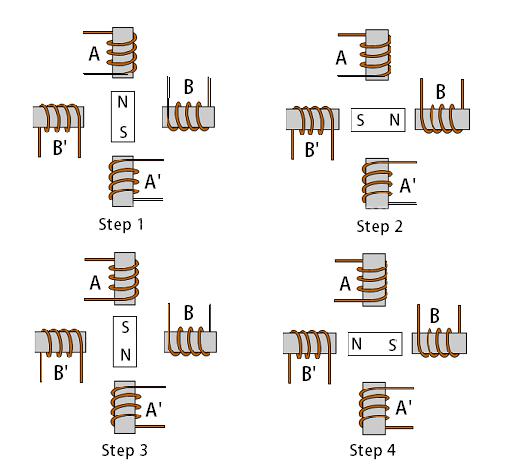


Figure 2: Full Step, 1 Phase ON

Pinout Diagram

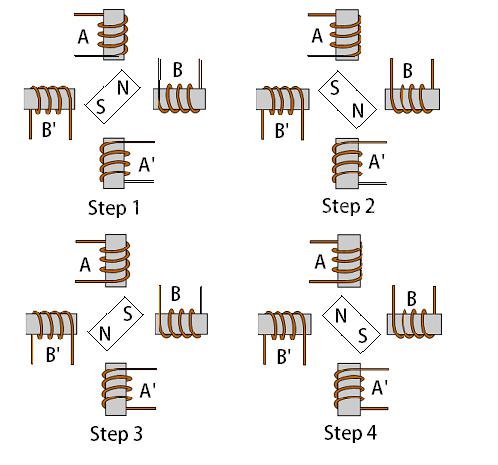


Figure 3: Full Step, 2 Phase ON

Algorithm

Full Step (Anti Clockwise)

1. Import appropriate libraries into the program.

import Adafruit\_BBIO.GPIO as GPIO import time

1. Set all the pins on the kit to OUT mode. GPIO.setup("P9\_11",GPIO.OUT)
2. Loop Begins
3. Set pin P9 12 HIGH and pins P9 11, P9 13, P9 14 LOW and sleep for 0.5 secs.
4. Set pin P9 13 HIGH and pins P9 11, P9 12, P9 14 LOW and sleep for 0.5 secs.
5. Set pin P9 14 HIGH and pins P9 11, P9 12, P9 13 LOW and sleep for 0.5 secs.
6. Set pin P9 11 HIGH and pins P9 12, P9 13, P9 14 LOW and sleep for 0.5 secs.

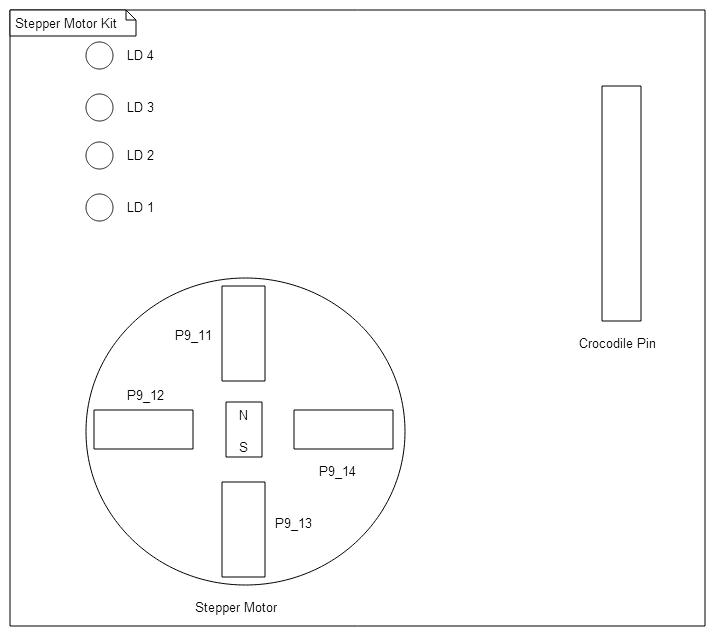


Figure 5: Pinout Diagram

GPIO.output("P9\_11", GPIO.LOW)

GPIO.output("P9\_12", GPIO.HIGH)

GPIO.output("P9\_13", GPIO.LOW)

GPIO.output("P9\_14", GPIO.LOW)

time.sleep(0.5)

GPIO.output("P9\_11", GPIO.LOW)

GPIO.output("P9\_12", GPIO.LOW)

GPIO.output("P9\_13", GPIO.HIGH)

GPIO.output("P9\_14", GPIO.LOW)

time.sleep(0.5)

GPIO.output("P9\_11", GPIO.LOW)

GPIO.output("P9\_12", GPIO.LOW)

GPIO.output("P9\_13", GPIO.LOW)

GPIO.output("P9\_14", GPIO.HIGH)

time.sleep(0.5)

GPIO.output("P9\_11", GPIO.HIGH)

GPIO.output("P9\_12", GPIO.LOW)

GPIO.output("P9\_13", GPIO.LOW)

GPIO.output("P9\_14", GPIO.LOW)

time.sleep(0.5)

**Input Output**

|  |  |
| --- | --- |
| Input (Steps) | Output (Moment) |
|  |  |
| Full Step | 13.84 |
|  |  |
|  |  |

**Conclusion**

Thus, studied Stepper motor program and its practical implementation using Beaglebone Black Board in python.

Assignment Number: - C-2

**Problem Statement**

Write an application using Raspberry-Pi /Beagle board to control the operation of a hardware simulated traffic signal.

**Objective**

* To understand the architecture of Beaglebone Black.
* To understand how to write programs for Beaglebone Black in Python.

**Theory**

Architecture of ARM Cortex A8

The ARM Cortex-A8 is a 32-bit processor core licensed by ARM Holdings implementing the ARMv7-A architecture. It supports provides full virtual memory capabilities. The features of the processor include full implementation of the ARM architecture v7-A instruction set ,congurable 64-bit or 128-bit high-speed Advanced Microprocessor Bus Architecture (AMBA) with Advanced Extensible Interface (AXI) for main memory interface supporting multiple outstanding transactions ,Memory Management Unit (MMU) and separate instruction and data Translation Look-aside Buers (TLBs) etc. The main components of the processor are as below.

Instruction Fetch

The instruction fetch unit predicts the instruction stream, fetches instructions from the L1 instruction cache, and places the fetched instructions into a buer for consumption by the decode pipeline.

Instruction decode

The instruction decode unit decodes and sequences all ARM and Thumb-2 instructions including debug control coprocessor, CP14, instructions and system control coprocessor, CP15, instructions.

Instruction execute

The instruction execute unit consists of two symmetric Arithmetic Logical Unit (ALU) pipelines, an address generator for load and store instructions, and the multiply pipeline.

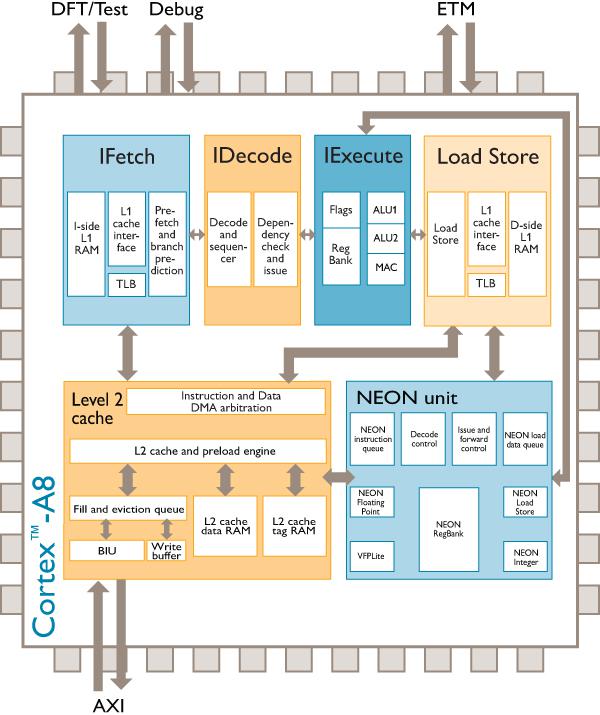


Figure 2: ARM Cortex A8 Block Diagram

Load/store unit

The load/store unit encompasses the entire L1 data side memory system and the integer load/store pipeline. This includes:

1. the L1 data cache.
2. the data side TLB.
3. the integer store bu er.
4. the NEON store bu er.
5. the integer load data alignment and formatting.

L2 cache

The L2 cache unit includes the L2 cache and the Bu er Interface Unit (BIU). It services L1 cache misses from both the instruction fetch unit and the load/store unit.

NEON

The NEON unit includes the full 10-stage NEON pipeline that decodes and executes the Advanced SIMD media instruction set.

ETM

The ETM (Embedded Trace Macrocell) unit is a non-intrusive trace macrocell that lters and compresses an instruction and data trace for use in system debugging and system pro ling.

Python Programming

Adafruit BBIO.GPIO Library

A module to control BeagleBone IO channels.Library to provide a cross-platform GPIO interface on the Raspberry Pi and Beaglebone Black using the RPi.GPIO and Adafruit BBIO libraries.

The library is currently in an early stage, but you can see how its used in the Adafruit Nokia LCD library to write Python code that is easily portable between the Raspberry Pi and Beaglebone Black.

Setup

To setup a digital pin as an output, set the output value HIGH,

import Adafruit\_BBIO.GPIO as GPIO

GPIO.setup("P8\_10", GPIO.OUT)

GPIO.output("P8\_10", GPIO.HIGH)

GPIO.cleanup()

Pins can be set as inputs as follows:

import Adafruit\_BBIO.GPIO as GPIO

GPIO.setup("P8\_14", GPIO.IN)

Time Library

This module provides various time-related functions. Suspend execution of the current thread for the given number of seconds. The argument may be a oating point number to indicate a more precise sleep time. The actual suspension time may be less than that requested because any caught signal will terminate the sleep() following execution of that signals catching rou-tine. Also, the suspension time may be longer than requested by an arbitrary amount because of the scheduling of other activity in the system.

Example: import time

time.sleep( 5 )

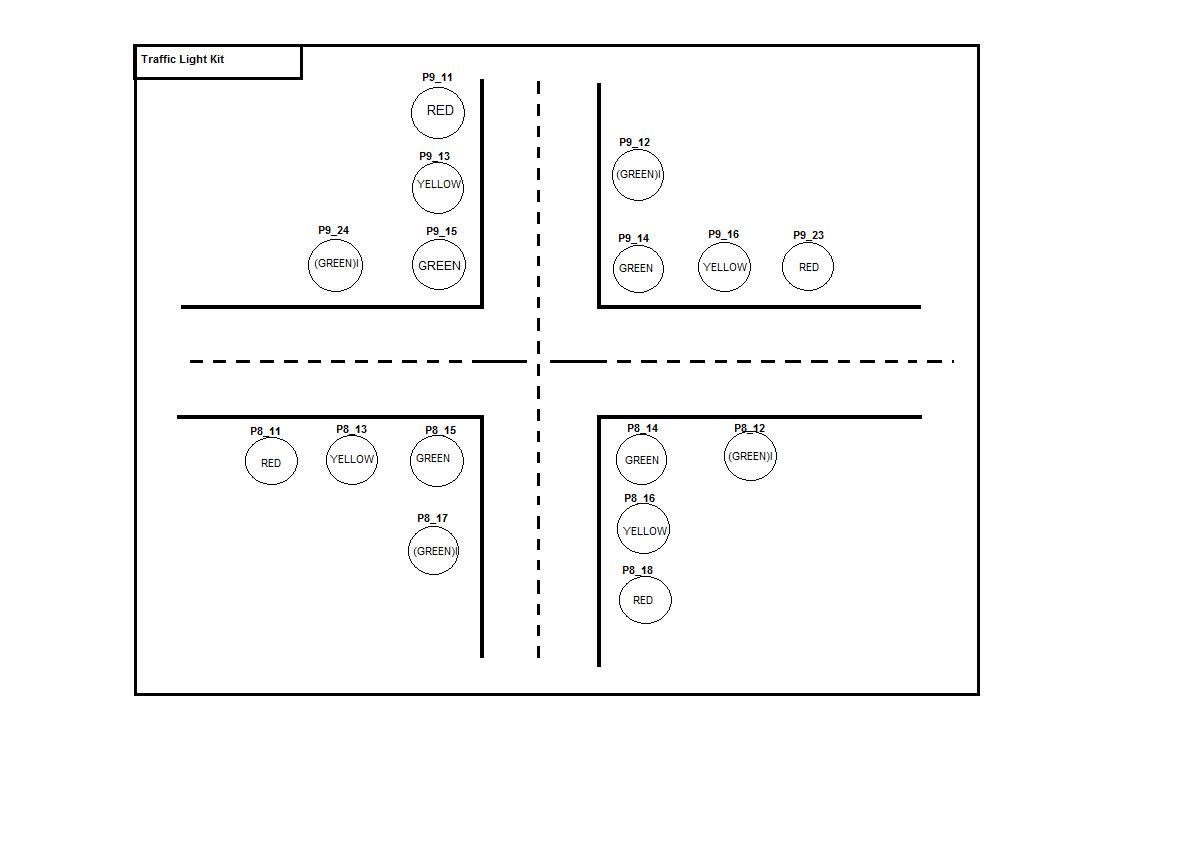


Figure 3: Pinout Diagram

Algorithm

1. Import appropriate libraries into the program.

import Adafruit\_BBIO.GPIO as GPIO import time

1. Set the count variable for loop iteration. count=1
2. Set all the pins on the kit to OUT mode. GPIO.setup("P9\_11",GPIO.OUT)
3. Loop Begins
   * Set the Green LED of vertical road to LOW and sleep for 5 secs.

Set the Green LED and Yellow LEDs of vertical road to HIGH and LOW resp and sleep for 2secs.

Set the Green LED, Yellow and Red LEDs of vertical road to HIGH, HIGH and LOW resp also set the Green LED of horizontal road to LOW and sleep for 5 secs.

Set the Green LED and Yellow LEDs of horizontal road to HIGH and LOW resp and sleep for 2secs.

Set the Green LED, Yellow and Red LEDs of horizontal road to HIGH, HIGH and LOW resp.

Example:

GPIO.output("P8\_18",GPIO.LOW) GPIO.output("P9\_11",GPIO.LOW)

GPIO.output("P8\_14",GPIO.HIGH) GPIO.output("P9\_15",GPIO.HIGH)

GPIO.output("P8\_11",GPIO.HIGH) GPIO.output("P9\_23",GPIO.HIGH)

GPIO.output("P8\_17",GPIO.LOW)

GPIO.output("P9\_12",GPIO.LOW)

GPIO.output("P8\_12",GPIO.HIGH)

GPIO.output("P9\_24",GPIO.HIGH)

time.sleep(5)

//Active Low

1. Increment count count =count +1
2. Loop Ends

**Input Output**

Input Output

Glowing Traffic Lights

**Conclusion**

Thus, studied Traffic light program and its practical implementation using Beaglebone Black Board in python.

Assignment Number: - C-3

Problem Statement

Write an application using Raspberry-Pi /Beagle board to control the operation of a hardware simulated lift elevator

Objective

* To understand the operations of LIFT.
* To understand how to write programs for Beaglebone Black in Python.

Pinout Diagram

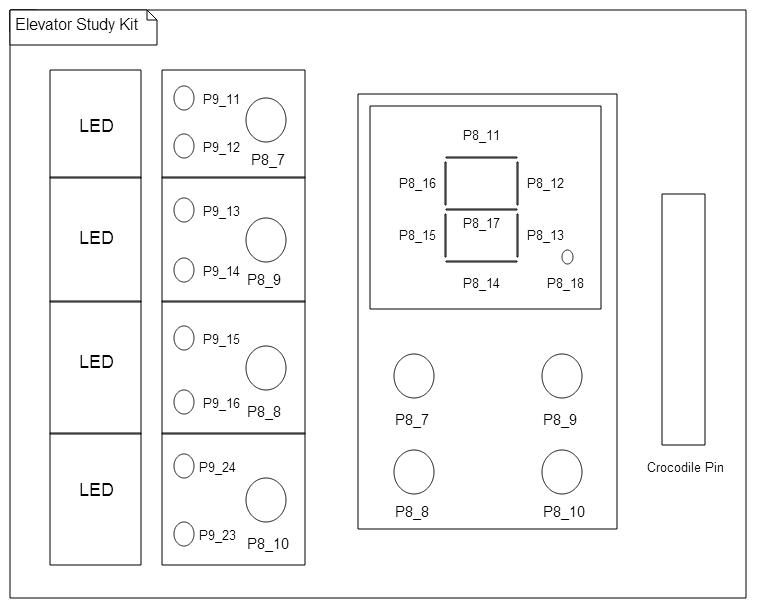


Figure 1: Pinout Diagram

**Algorithm**

1. Import appropriate libraries into the program.

Ex:

import Adafruit\_BBIO.GPIO as GPIO import time

1. Set the pins on the kit to IN/OUT mode.

Ex:

GPIO.setup("P8\_7",GPIO.IN) GPIO.setup("P8\_11",GPIO.OUT)

1. Set Flags.

Ex:

lg=1

lf=0

ls=0

lt=0

1. De ne functions pzero(),pone(),ptwo(),pthree() to print zero, one, two, three respectively i.e to display zero, one, two, three on the seven seg-ment display.

Ex:

def pzero():

GPIO.output("P8\_11",GPIO.LOW) GPIO.output("P8\_12",GPIO.LOW) GPIO.output("P8\_13",GPIO.LOW) GPIO.output("P8\_14",GPIO.LOW) GPIO.output("P8\_15",GPIO.LOW) GPIO.output("P8\_16",GPIO.LOW)

GPIO.output("P8\_17",GPIO.HIGH) GPIO.output("P8\_18",GPIO.HIGH)

1. De ne function led() which glows appropriate LEDs(green, yellow).

Ex:

def led(str) :

GPIO.output("P9\_11",GPIO.LOW)

GPIO.output("P9\_12",GPIO.LOW)

GPIO.output("P9\_13",GPIO.LOW)

GPIO.output("P9\_14",GPIO.LOW)

GPIO.output("P9\_15",GPIO.LOW)

GPIO.output("P9\_16",GPIO.LOW)

GPIO.output("P9\_24",GPIO.LOW)

GPIO.output("P9\_23",GPIO.LOW)

GPIO.output(str,GPIO.HIGH)

1. Initiate loop
   1. If input at pin P8 10 is 0, (lift is called at ground oor) then
      1. if ag lg is set then display zero (call pzero() function).
      2. else if the ag lf is set then display zero,set pin P9 23 to high(call led() function and send pin as parameter), set lf to 0 and lg to 1.
      3. else if the ag ls is set then display one,set pin P9 16 to high, sleep for 1 sec, display zero ,set pin P9 23 to high ,set ls to 0 and lg to 1.
      4. else if the ag lt is set then display two,set pin P9 14 to high, , sleep for 1 sec, display one,set pin P9 16 to high, sleep for 1 sec, display zero,set pin P9 23 to high, set lt to 0 and lg to 1.
   2. If input at pin P8 8 is 0, (lift is called at rst oor) then
      1. if ag lg is set then display one,set pin P9 15 to high, set lg to 0 and lf to 1.
      2. else if the ag lf is set then display one.
      3. else if the ag ls is set then display one,set pin P9 16 to high, set ls to 0 and lf to 1.
      4. else if the ag lt is set then display two ,set pin P9 14 to high, sleep for 1 sec, display one, set pin P9 16 to high,set lt to 0 and lf to 1.
   3. If input at pin P8 9 is 0, (lift is called at second oor) then
   4. if ag lg is set then display one,set pin P9 15 to high,sleep for 1 sec, display two,set pin P9 13 to high, set lg to 0 and ls to 1.
   5. else if the ag lf is set then display two,set pin P9 13 to high, set lf to 0 and ls to 1.
   6. else if the ag ls is set then display two.
   7. else if the ag lt is set then display two,set pin P9 14 to high ,set lt to 0 and ls to 1.
2. If input at pin P8 7 is 0, (lift is called at third oor) then
   1. if ag lg is set then set pin P9 24 to high, display one,set pin P9 15 to high, sleep for 1 sec, display two,set pin P9 13 to high, sleep for 1 sec, display three,set pin P9 11 to high, set lg to 0 and lt to 1.
   2. else if the ag lf is set then display two,set pin P9 13 to high,sleep for 1 sec, display three, set pin P9 11 to high, set lf to 0 and lt to 1.
   3. else if the ag ls is set then display three,set pin P9 11 to high, set ls to 0 and lt to 1.
   4. else if the ag lt is set then display three.

Ex:

if (GPIO.input("P8\_10")==0 ): #0(Ground floor)

if lg==1:

pzero()

time.sleep(1)

elif lf==1:

pzero()

led("P9\_23")

lf=0

lg=1

elif ls==1:

pone()

led("P9\_16")

time.sleep(1)

pzero()

led("P9\_23")

ls=0

lg=1

elif lt==1:

ptwo()

led("P9\_14")

time.sleep(1)

pone()

led("P9\_16")

time.sleep(1)

pzero()

led("P9\_23")

time.sleep(1)

lt==0

lg==1

7. Loop ends.

**Input and Output**

Sr. No. Input Output

* P8 10 Ground Floor
* P8 8 First Floor
* P8 9 Second Floor
* P8 7 Top Speed

**Conclusion**

Thus, studied the program to simulate operations of LIFT and its practical implementation using Beaglebone Black Board in python.

**Assignment Group D-1**

**Problem Statement**

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

**Objective**

To understand the concept of socket programming.

To understand how to write programs for Beaglebone Black in Python.

**Theory**

Socket

Sockets are the endpoints of a bidirectional communications channel. A socket is bound to a port number so that the TCP layer can identify the ap-plication that data is destined to be sent to.Sockets may communicate within a process, between processes on the same machine, or between processes on dierent continents.Sockets may be implemented over a number of dierent channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specic classes for handling the common transports as well as a generic interface for handling the rest. A socket address is the combina-tion of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particularextension. Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread.

Working

Normally, a server runs on a specic computer and has a socket that is bound to a specic port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server’s machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system. If everything goes

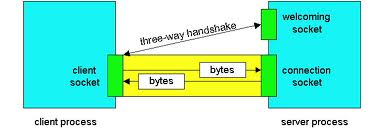


Figure 1: Client Server Connection

well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

Socket Functions

1. Socket:

Ex:

s = socket.socket (socket\_family, socket\_type, protocol=0)

**Where :-**

socket family: = This is either AF UNIX or AF ~~I~~NET.

socket type = This is either SOCK ~~S~~TREAM or SOCK DGRAM.

protocol=This is usually left out, defaulting to 0.

2. Bind:

Ex:socket.bind(IPaddress,Port number)

This method binds address (hostname, port number pair) to socket. The socket must not already be bound. (The format of address depends on the address family).

3. Listen:

Ex:socket.listen(backlog)

Listen for connections made to the socket. The backlog argument spec-ies the maximum number of queued connections and should be at least 0; the maximum value is system-dependent (usually 5), the minimum value is forced to 0.

4. Accept:

Ex:conn, address = socket.accept()

Accept a connection. The socket must be bound to an address and listening for connections. The return value is a pair (conn, address) where conn is a new socket object usable to send and receive data on the connection, and address is the address bound to the socket on the other end of the connection.

5. Connect:

Ex:socket.connect(IP address, Port address)

Connect to a remote socket at address. (The format of address depends on the address family).

6. Recieve:

Ex:socket.recv(bufsize[], flags)

Receive data from the socket. The return value is a string representing the data received. The maximum amount of data to be received at once is specied by bufsize. Optional argument ags are by default set to zero. Note: For best match with hardware and network realities, the value of bufsize should be a relatively small power of 2, for example, 4096.

7. Send:

Ex:socket.send(string[], flags)

Send data to the socket. The socket must be connected to a remote socket. The optional ags are by default set to 0. Returns the number of bytes sent. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data.

8. Close:

Ex:socket.close()

Close the socket. All future operations on the socket object will fail. The remote end will receive no more data (after queued data is pushed). Sockets are automatically closed when they are garbage-collected. Note: close() releases the resource associated with a connection but does not necessarily close the connection immediately. If you want to close the connection in a timely fashion, call shutdown() before close().

Python Programming Socket Library

This module provides various socket-related functions. It provides access to the BSD socket interface. It is available on all modern Unix systems, Win-dows, MacOS, and probably additional platforms.The socket module con-sists of the socket() function, which is used for socket initiation/creation i.e.

socket.socket(). The socket module also provides several other methods.It supports various protocols for communication, viz., TCP (Transmission Con-trol Protocol), UDP (User Datagram Protocol), etc.

Ex.

import socket

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

**Algorithm**

**Server Side**

1. Import socket library into the program.

import socket

2. Set the IP address and port number of Server.

TCP\_IP = ’192.168.7.2’ TCP\_PORT = 5004

3. Set Buer size.

BUFFER\_SIZE = 20

4. Create a socket using socket() function.

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

5. Bind the server socket using bind() function.

s.bind((TCP\_IP, TCP\_PORT))

6. Let server listen to clients willing to connect to it using listen() function.

s.listen(1)

7. Accept an incoming connection request using accept() function.

conn, addr = s.accept()

8. Once request accepted, initiate loop.

(a) Receive data from client using recv() function and display it. If no data received print appropriate message.

data = conn.recv(BUFFER\_SIZE)

(b) Input the data to be sent to client and send it using send() func-tion.

conn.send(data1)

9. Loop ends.

10. Close connection.

conn.close()

**Client Side**

1. Import socket library into the program.

import socket

2. Set the IP address and port number.

TCP\_IP = ’192.168.7.2’ (IP address of server) TCP\_PORT = 5004

3. Set Buer size.

BUFFER\_SIZE = 20

4. Create a socket using socket() function.

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

5. Connect to server socket using connect() function.

s.connect((TCP\_IP, TCP\_PORT))

6. Initiate loop.

(a) Input the data to be sent to server and send it using send() func-tion.

s.send(msg)

(b) Receive data from server using recv() function and display it.If no data received print appropriate message.

data = s.recv(BUFFER\_SIZE)

7. Loop ends.

8. Close connection.

s.close()

Input and Output

|  |  |  |
| --- | --- | --- |
|  | Input | Output |
| Server | Client Message | Server Message |
| Client | Server Message | Client Message |

**Conclusion**

Thus, studied Network based program(Client Server program) and its practical implementation using Beaglebone Black Board in python.

**Assignment Number: D-2**

**Problem Statement:**

Create a simple web interface for Raspberry-pi/Beagle board to control the connected LEDs remotely through the interface.

**Theory:**

**Step 1:** **Components Required**

The components required for this project are :

1. Raspberry Pi 3

2. LED

3. 270 ohm resistor

The raspberry pi should be provided with an SD card loaded with an appropriate OS(preferably Raspbian) and should be connected to a monitor via HDMI to VGA converter or to a laptop via SSH. Apart from the above components a router with internet connection should also be present, with the raspberry pi being connected to it.

**Step 2: Connections:**

The connections in this project are very simple. Connect the positive pin of LED to GPIO 17 pin and the negative to a 270 ohm resistor, the other side of which is connected to GND pin.

**Step 3:** **Installing WiringPi Library**

WiringPi is basically a GPIO interface library for Raspberry Pi. There are 2 methods to install wiringPi library.

(All the commands given below are to be executed on the Pi's terminal).

**Pinout Diagram:**

Method 1

1. Make sure your Pi is up to date with latest versions of Raspbian by :

sudo apt-get update

2. Install git by the command: sudo apt-get install git-core

3. Obtain WiringPi using git by: git clone git://git.drogon.net/wiringPi

4. To build/install WiringPi library, cd wiringPi  
./build

Method 2

Try out this method only if method 1 doesn't work out.

<https://git.drogon.net/?p=wiringPi;a=summary>

Click the above URL and check for the link marked snapshot at the right hand side. Click the top one which will download a tar.gz file like wiringPi-xxxxxxx.tar.gz (xxxxxxx represents a combination of letters and numbers which is unique for each download).

Then open the terminal and type these commands:

tar xfz wiringPi-xxxxxxx.tar.gz  
cd wiringPi-xxxxxxx ./build

Thus the WiringPi library will be installed by one of the above methods. For more details about WiringPi library, please visit [http://wiringpi.com/.](http://wiringpi.com/) To make sure Wiring Pi is installed and works properly, run the gpio -v command; it should return the current version of Wiring Pi along with the basic Raspberry Pi info.

**Step 4:** **Installing a Web Server**

Apache is a popular web server application you can install on the Raspberry Pi to allow it to serve web pages. On its own, Apache can serve HTML files over HTTP, and with additional modules can serve dynamic web pages using scripting languages such as PHP.

In our project we are using an HTTP server and its PHP extension. MySQL database is not used here.

First install Apache HTTP server and its PHP extension by:

sudo apt-get install apache2 php libapache2-mod-php

In order to test if the Apache server is working properly, navigate to the browser and type your Pi's IP address.

(To get Raspberry Pi's IP address type ifconfig on the terminal. Next to the wlan0 entry you will see inet addr: 192.168.x.x which is the IP address of the Raspberry Pi.)

Now we should test whether its PHP extension is working. The above "It works!" html page is present in "/var/www/" directory as index.html. Now, delete this html file and create a new PHP file index.php. Then type the php code below :

phpinfo();

Save it and refresh the page in your browser. A long page with lots of information about PHP will appear. Thus the PHP extension is installed properly. If any of the pages do not appear try reinstalling apache server and its PHP extension.

**Step 5**: **Start Coding**

We will be coding in PHP to control the GPIO pins, which is embedded in an HTML page. GPIO pins of the Raspberry Pi can be controlled by PHP using shell\_exec() function. This function executes command via shell and returns the complete output as a string.

Now delete the code in index.php file and insert PHP code to control GPIO pins inside body of HTML code. If you are unable to edit the index.php file due to some administrative problems, navigate to "/var/www/" directory using cd /var/www/ command and type the code - sudo chmod777 index.php. Now you will be able to edit the php file.

First create a form in HTML consisting of 3 buttons - ON, OFF and BLINK .

Then set the GPIO 17 pin as output with the help of WiringPi library using the statement:

shell\_exec("/usr/local/bin/gpio -g mode 17 out");

**Algorithm:**

Check for the values returned by the button and accordingly make the GPIO pin HIGH/LOW using if else-if statements.

if(isset($\_GET['off']))  
{

echo "LED is off";

shell\_exec("/usr/local/bin/gpio -g write 17 0");

}

else if(isset($\_GET['on']))

{

echo "LED is on";

shell\_exec ("/usr/local/bin/gpio -g write 17 1");

}

else if(isset($\_GET['blink']))  
{

echo "LED is blinking";

for($x = 0;$x<=4;$x++)

{

shell\_exec("/usr/local/bin/gpio -g write 17 1");

sleep(1);

shell\_exec("/usr/local/bin/gpio -g write 17 0");

sleep(1);**}}**

**Conclusion:**

We have successfully implemented the web interface for Raspberry-pi to control the connected LEDs remotely through the interface.

**CODE**

<html>

<head>

</head>

<body>

<form method="get" action="index.php">

<input="submit" value="OFF" name="off">

<input="submit" value="ON" name="on">

<input="submit" value="BLINK" name="blink">

</form>

<?php

shell\_exec("/usr/local/bin/gpio -g mode 17 out");

if(isset($\_GET['off']))

{

echo "LED is off";

shell\_exec("/usr/local/bin/gpio -g write 17 0");

}

else if(isset($\_GET['on']))

{

echo "LED is on";

shell\_exec("/usr/local/bin/gpio -g write 17 1");

}

else if(isset($\_GET['blink']))

{

echo "LED is blinking";

for($x = 0;$x<=4;$x++)

{

shell\_exec("/usr/local/bin/gpio -g write 17 1");

sleep(1);

shell\_exec("/usr/local/bin/gpio -g write 17 0");

sleep(1);

}

}

?>

</body>

</html>