Nithin Kurup U G
Assistant Professor
Dept: of CSE

INTERNET OF THINGS TECHNOLOGY

Subject Code- 15CS81

Credits – 04

Module – 5

IoT Physical Devices and Endpoints - Arduino UNO: Introduction to Arduino, Arduino UNO, Installing the Software, Fundamentals of Arduino Programming. IoT Physical Devices and Endpoints - RaspberryPi: Introduction to RaspberryPi, About the RaspberryPi Board: Hardware Layout, Operating Systems on RaspberryPi, Configuring RaspberryPi, Programming RaspberryPi with Python, Wireless Temperature Monitoring System Using Pi, DS18B20 Temperature Sensor, Connecting Raspberry Pi via SSH, Accessing Temperature from DS18B20 sensors, Remote access to RaspberryPi, Smart and Connected Cities, An IoT Strategy for Smarter Cities, Smart City IoT Architecture,Smart City Security Architecture, Smart City Use-Case Examples.

Text Book:

1. David Hanes, Gonzalo Salgueiro, Patrick Grossetete, Robert Barton, Jerome Henry,**"IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things",** 1stEdition, Pearson Education (Cisco Press Indian Reprint). (**ISBN:** 978-9386873743).

2. Srinivasa K G, **"Internet of Things",**CENGAGE Leaning India, 2017

Reference Books:

1. Vijay Madisetti and ArshdeepBahga, **"Internet of Things (A Hands-on-Approach)",** 1stEdition, VPT, 2014. (**ISBN:** 978-8173719547)

2. Raj Kamal, **"Internet of Things: Architecture and Design Principles",** 1st Edition, McGraw Hill Education, 2017. (**ISBN:** 978-9352605224)

Module- 5

5) Introduction to ARDUINO

- Arduino is an open-source advancement prototyping (development model) platform which depends on simple to utilize equipment and programming.
- Instructions to the microcontroller are given by the use of Arduino programming.
- Arduino software(IDE-Integrated improvement environment)
- The Arduino is a small computer that you can program to read information from the world around you and to send commands to the outside world.
- Arduino is a tiny computer that you can connect to electrical circuits.
- The brain of this board (Arduino Uno) is an ATmega328p chip (Micro controller) where you can store your programs that will tell your Arduino what to do.

5.1) **Why Arduino?**

1) Arduino is a open source product , software/hardware which is accessible and flexible to customers.
2) Arduino is flexible because of offering variety of digital and analog pins , SPI (Serial Peripheral Interface) and PWM
3) The **PWM pins** are used for giving the desired analog output .They are used to set the LED brightness or to run Stepper or Servo Motor or anything which require analog inputs 3, 5, 6, 9, 11 (PWM)
4) Serial Peripheral Interface (SPI) :- is an interface bus commonly used to send data between microcontrollers and small peripherals such as sensors, and SD cards.
5) Arduino is easy to use, connected to a computer via a USB and communicates using serial protocol .
6) Inexpensive around 500 rupees per board with free authoring software.
7) Ardunio has growing online community where lots of source code is available for use .
8) Ardunio is Cross-platform, which can work on windows,Mac or Linux platforms.
9) Ardunio follows simple, clear programming environment as C-language.

5.1.2) Which Arduino?

- In the ten years since Arduino was released , hundreds of “Arduino boards” are available in the market serving every kind of purpose.
- We focus on Arduino UNO .
- Some of the Boards from Arduino family are given below.

- Arduino mega is a big sister to the UNO with more memory and pins with a different chip the Atmega2560,useful when your project doesn”t fits in an UNO.
- Arduino Micro is bit smaller with a chip Atmega32u4 that can act like a keyboard or mouse which does its task with native USB.
- Its slim with downward pins which can be plugged into a breadboard.
- The Arduino MKR1000 is a little like an Arduino Micro but has a more powerful 32-bit ATSAM ARM chip and built-in WiFi.
- A great upgrade for when you want to do internet of Things projects.
- Flora is an Arduino compatible from Adafruit which is a round wearable which can be sewed (attach) into clothes.

5.2) Exploring ARDUINO UNO Learning Board.

- ❖ Microcontroller:- The ATmega328p is the Arduino brain.Everything on the Arduino board is meant to support this microcontroller
- ❖ Digital pins- 0-13
- ❖ For input or output
    - Apply 5v (HIGH)
    - 0V (LOW)

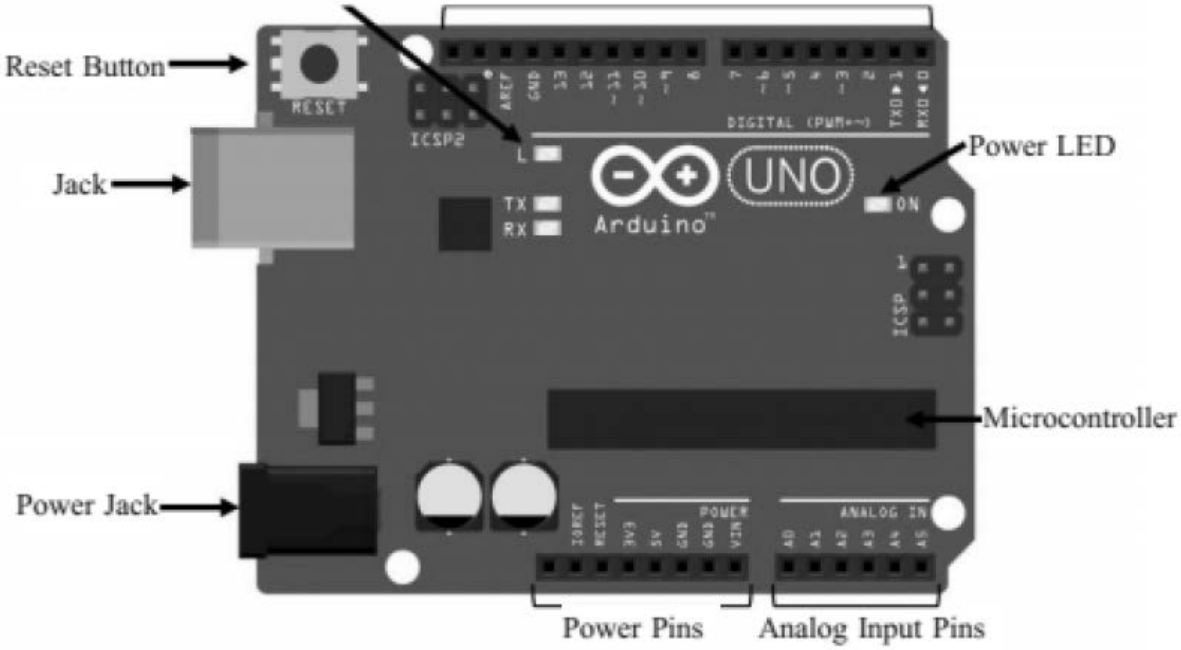


Figure 5-1: Arduino Uno Learning Board

- PWM pins: These are digital pins marked with a ~ (pins 11, 10, 9, 6, 5 and 3). PWM stands for "pulse width modulation" and allows to make digital pins output "fake" varying amounts of voltage. You'll learn more about PWM later.
- TX and RX pins: digital pins 0 and 1. The T stands for "transmit" and the R for "receive". Arduino uses these pins to communicate with the computer. Avoid using these pins, unless you're running out of pins.
- LED attached to digital pin 13: This is useful for an easy debugging of the Arduino sketches.
- TX and RX pins: these pins blink when there are information being sent between the computer and the Arduino.
- Analog pins: the analog pins are labeled from A0 to A5 and are most often used to read analog sensors. They can read different amounts of voltage between 0 and 5V. Additionally, they can also be used as digital output/input pins like the digital pins.
- Power pins: The Arduino has 3.3V or 5V supply, which is really useful since most components require 3.3V or 5V. The pins labelled as "GND" are the ground pins.
- Reset button: when you press that button, the program that is currently being run in your Arduino will start from the beginning. You also have a Reset pin next to the power pins that acts as reset button. When you apply a small voltage to that pin, it will reset the Arduino.
- Power ON LED: will be on since power is applied to the Arduino.
- USB jack: Connecting a male USB A to male USB B cable is how you upload programs from your computer to your Arduino board. This also powers your Arduino.
- Power jack: The power jack is where you connect a component to power up your Arduino (recommended voltage is 5V). There are several ways to power up your Arduino: rechargeable batteries, disposable batteries, wall-warts and solar panel.

5.2.1) Things that Arduino can do

- The simplest thing you can control with your Arduino is an LED.
- You can also display a message in a display, like the LCD display.
- You can also control DC or servo motors.
- You can also Read data from the outside world
- Motion sensor: The motion sensor allows you detect movement.
- Light sensor: this allows you to "measure" the quantity of light in the outside world.
- Humidity and temperature sensor: this is used to measure the humidity and temperature.
- Ultrasonic sensor: this sensor allows to determine the distance to an object through sonar.
- Shields – an extension of the Arduino
- Shields are boards that will expand the functionalities of your Arduino. You just need to plug them over the top of the Arduino. There are countless types of shields to do countless tasks.

5.3) Installing the Software(Arduino IDE)

- The Arduino IDE (Integrated Development Environment) is where you develop your programs that will tell your Arduino what to do.
- You can load new programs onto the main chip, the ATmega328p,via USB using the Arduino IDE.
- Browse on the following link:.Select which Operating System you are using and download.

- https://www.arduino.cc/en/Main/Software

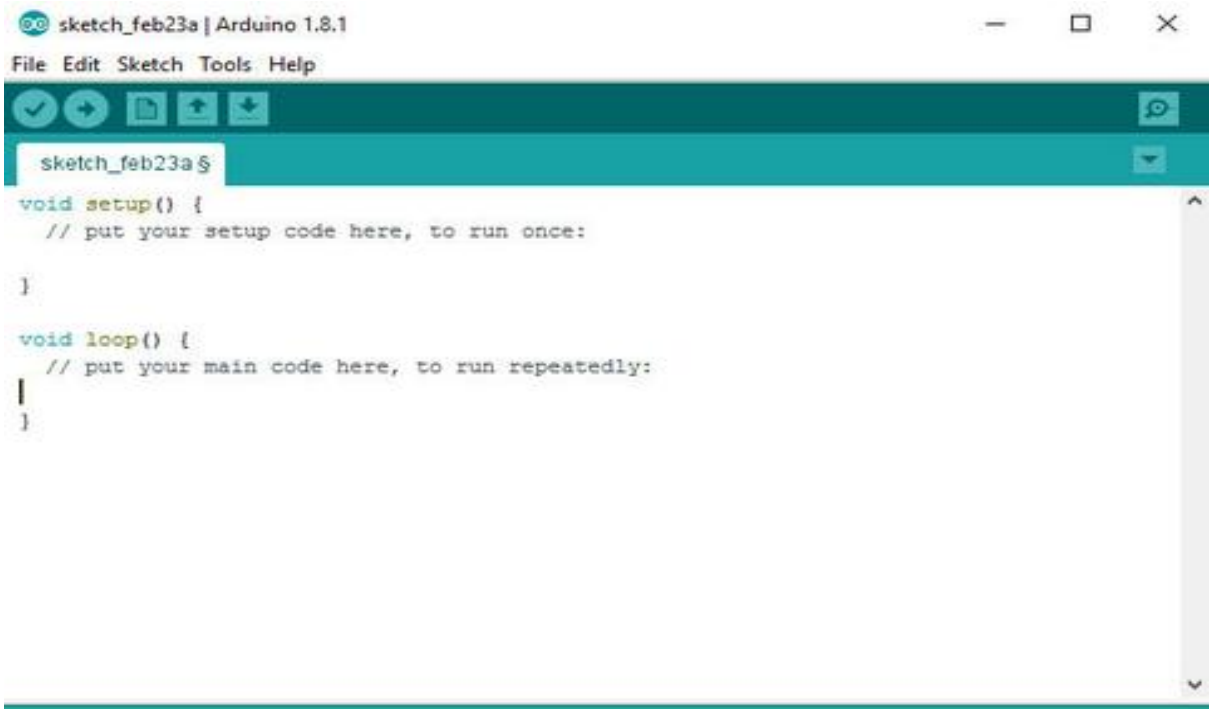


Figure 5.2 : Arduino IDE

5.3.1) Connecting Arduino Uno Learning Board.

- ❖ After connecting your Arduino with a USB cable, you need to make sure that Arduino IDE has selected the right board you are using.
- ❖ In our case , we are using Arduino Uno,so you should go to Tools>Board"Arduino/Genuino Uno"> Arduino/Genuino Uno as shown in figure 5- 3.
- ❖ Then you select the right serial port where your Arduino is connected to.
- ❖ Go to Tools>port and select the right port as shown in Figure 5-4 and Figure 5-5shows the layout of ArduinoIDE.

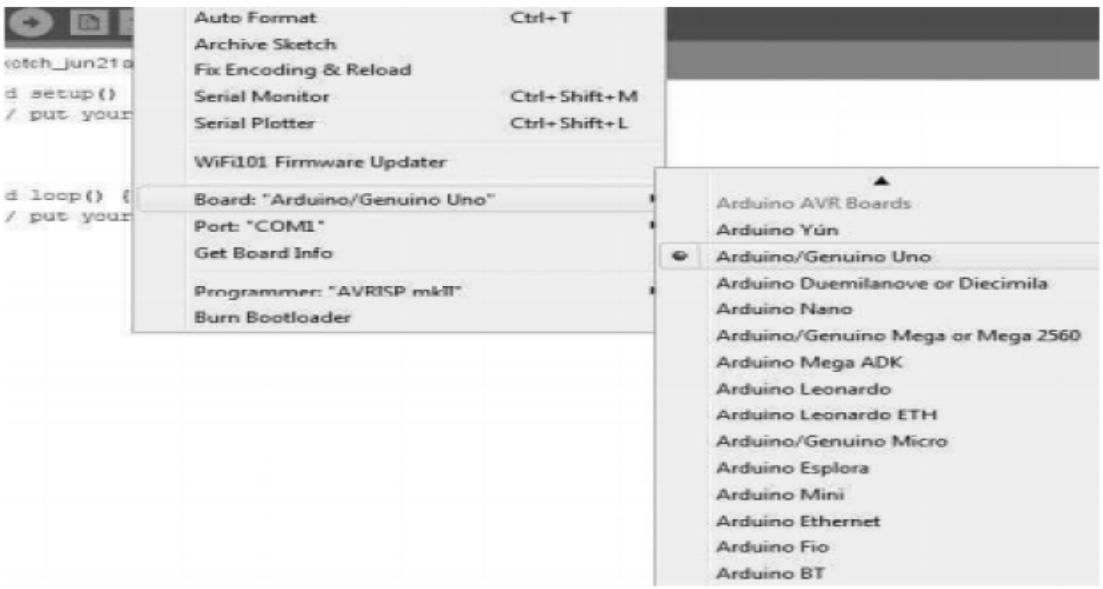


Figure 5.3 : Selecting the right Board

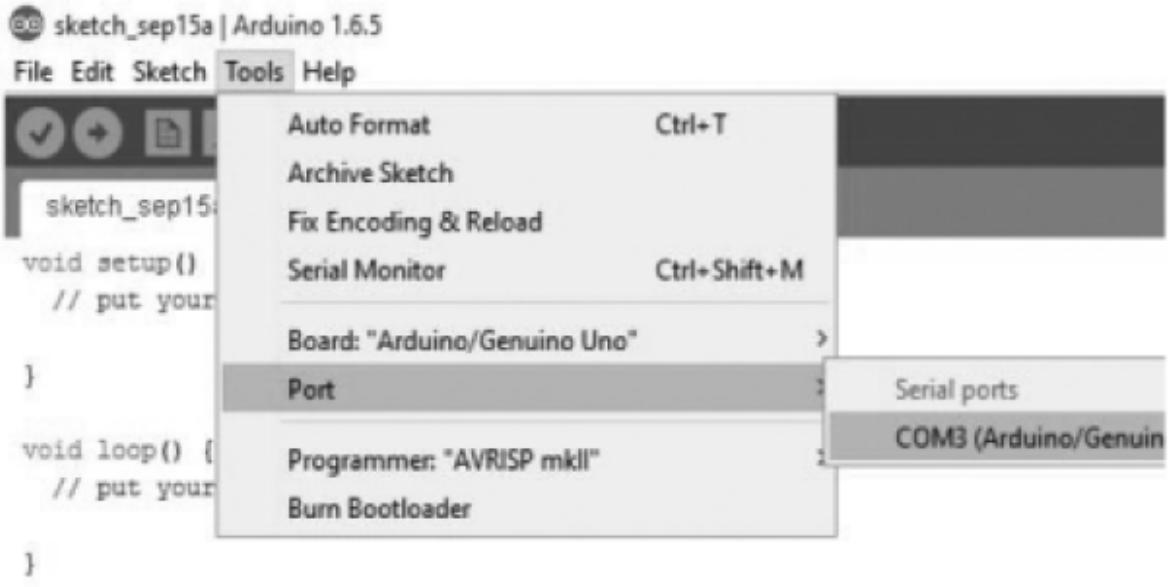


Figure 5.4 : Selecting the right port

Figure 5.5 : Layout of Arduino Uno IDE.

Breadboard for prototyping Arduino Uno Circuits

- In order to keep your circuit organized you need to use a breadboard, pictured below in Figure 5-6.
- The breadboard allows you to connect components together by plugging them into the little holes.

- The key is to understand how the holes are connected.

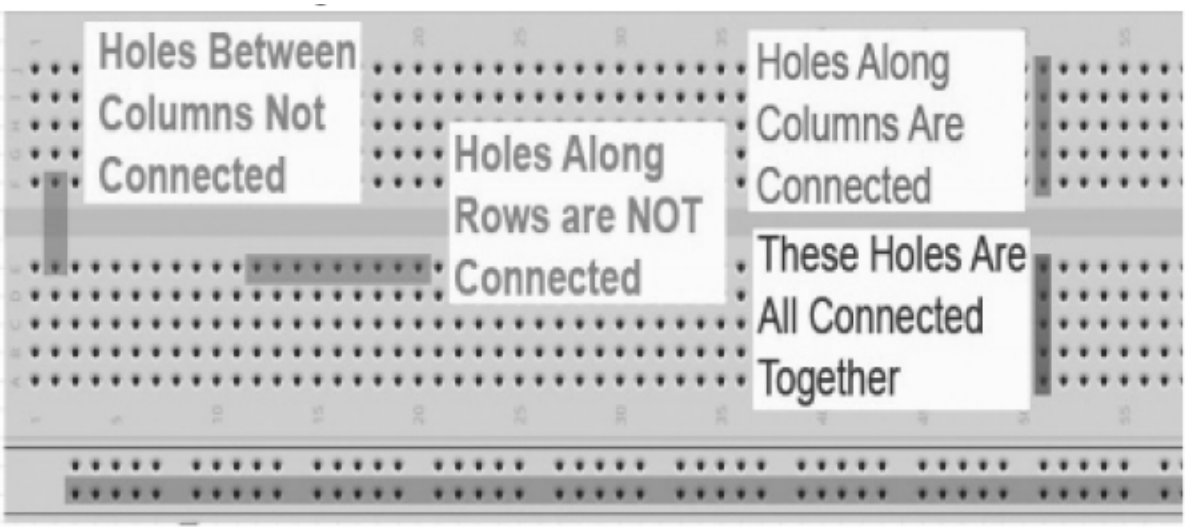


Figure 5.6 : Breadboard for prototyping Arduino Uno circuits.

Technical Specifications: Microcontroller Arduino UNO	A Tmega 328P
Operating Voltage	5V
Input Voltage (recommended)	7–12V
Input Voltage (limit)	6–20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pin	6
Analog Input Pins	6
DC Current per I/O Pin	20mA
DC Current for 3.3V Pin	50mA
Flash Memory	32 KB (A Tmega 328P) of which 0.5 KB used by bootloader
SRAM	2KB (A Tmega328P)
EEPROM	1 KB (A Tmega 328P)
Clock Speed	16MIIz

Table-1 Technical Specification of Arduino UNO

5.4) Fundamentals of Arduino Programming

In this section explains the basic structure of Arduino programming with respect to usage of variables, constants , control flow statement and finally the predefined functions used to read analog and digital inputs.

Structure	The structure of Arduino programming contains of two parts as shown below
	void setup() //Preparation function used to declare variables
	{ //First function that runs only one in the program
	Statement(s); //used to set pins for serial communication
	}
	void loop() //Execution block where instructions are executed repeatedly
	{ //this is the core of the Arduino programming
	Statements(); //Functionalities involve reading inputs, triggering outputs etc.
	}

void setup()	void setup()
	{
	pinMode(pin, INPUT); //'pin' configure as inpu
	}

void loop()	void loop() //After calling setup(),loop() function does its task
	{
	digitalWrite(pin, HIGH); //sets 'pin' ON
	delay(10000); //pauses for ten thousand mili second
	digitalWrite(pin, LOW); //sets 'pin' OFF
	delay(10000); //pauses for ten thousand mili second
	}

Functions	A function is a piece of code that has a name and set of statements executed when function is called. Functions are declared by its type followed with name of a function. Syntax: type functionName(parameters)

	{
	Statement(s);
	}
	Example:
	int delayvar()
	{
	int var; //create temporary variable var
	var=analogRead(potent); //read from potentiometer
	var=var/4; //convert the value of variable var
	return var; //return var
	}

{ } curly braces	They define beginning and end of function blocks, unbalanced braces may lead to compile errors.
semicolon	It is used to end a statement and separate elements of a program. Syntax: int x=14;
/*…..*/ block comments	Multiline comments begin with /* with a description of the block and ends with */. Syntax: /*This is an enclosed block of comments Use the closing comment to avoid errors*/
//line comments	Single line comment begins with // and ends with next instruction followed. Syntax: //This is a Single line comment

Variables	A variable is a way of storing value for later use in the program. A variable is defining by its value type as an int, long, float etc by setting a specified name and optionally assigning an initial value. A global variable can be seen in every part of the program which is declared at the beginning of program before setup() function. A local variable is defined inside a function in which it was declared.
	Example:
	int var; //variable 'var' visible to all functions
	void setup()
	{
	//nothing is required
	}
	void loop()

	{
	for(int local=0;local<5;)
	{
	local++;//variable 'local' is only visible within for loop
	}
	float local_f; //variable 'local_f' is visible only inside the loop
	}

Data Types	

Data type	Syntax	Range
Byte	byte x = 100;	0–255
Int	int y = 200;	32767 to–32768
Long	long var = 8000;	2147483647 to –2147483648
Float	float x = 3.14;	3.4028235E+38to – 3.4028235E+38
arrays	int myarray []={10,20,30,40}	Size depends on the data type associated with declaration.

<table>
<tr><td>Operators</td><td colspan="2"><table>
<tr><th>Operstor</th><th>Syntax and its usage</th></tr>
<tr><td>Arithmetic operators
(+,–,/,*)</td><td>x = x+5;
y–y–6;
z=z*2;
p=p/q;</td></tr>
<tr><td>Assignment operators
(=,++,—,+ =,=,*=,/=)</td><td>x++; //same as x=x+1
x+=y;//same asx=x+y
x–=y; //same asx=x–y
x*=y; //same as x=x*y
x/=y; //same as x=x/y</td></tr>
</table></td></tr>
<tr><td></td><td>Comparison operators
(==,!=,<,>,<=,<=)</td><td>x==y //xis equal to y
x!–y //xis not equal to y
x<y //xis less that y
x!–y // x is not equal to y</td></tr>
<tr><td></td><td>Logical operators
(&&,||,!)</td><td>x>2&&x<5 //Evaluates to true
only if both expression are true
x>2||y>2 //Evaluates to true if any one expression is true
!x>2 //true if only expression is false</td></tr>
</table>

Constants	Constants	Usage

	TRUE/FALSE	Boolean constants true=2 and false=0 defined in logic levels.
		if(b==TRUE)
		{
		//do something
		}

	INPUT/OUTPUT	Used with pinMode () function to define levels. pinMode (13,OUTPUT):
	HIGH/LOW	Used to define pin levels HIGH–1,ON,5 volts LOW –0,OFF, 0 volts Digital Write (13,HIGH);

Flow control Statements	

if	if(some_variable == value)
	{
	Statement(s); //Evaluated only if comparison results in a true value
	}

if…else	if(input==HIGH)
	{
	Statement(s); //Evaluated only if comparison results in a true value
	}
	else
	{
	Statement(s); //Evaluated only if comparison results in a false value
	}

for	for(initialization;condition;expression)
	{
	Dosomething; //Evaluated till condition becomes false
	}
	for(int p=0;p<5;p++) //declares p, tests if less than 5, increments by 1

	{
	digitalWrite(13,HIGH); //sets pin 13 ON
	delay(250); // pauses for ¼ second
	digitalWrite(13,LOW); //sets pin 13 OFF
	delay(250); //pause for ¼ second
	}

while	While loop executes until the expression inside parenthesis becomes false.
	while(some_variable ?? value)
	{
	Statement(s); //Evaluated till comparison results in a false value
	}

do...while	Bottom evaluated loop, works same way as while loop but condition is tested at the end of loop. do { Dosomething; }while(somevalue);

Digital and Analog input output pins and their usage		
Digital i/o	Methods	Usage

	pinMode (pin, mode)	Used in setup () method to configure pin to behave as INPUT/OUTPUT
		pinMode(pin, INPUT) //pin set to INPUT
		pinMode(pin, OUTPUT) // pin set to OUTPUT

	Digital Read (pin)	Read value from a specified pin with result being HIGH/LOW Val=digital Read(pin); // Val will be equal to input pin
	Digital Write(pin, value)	Outputs to HIGH/LOW on a specified pin. Digital Write(pin, HIGH); //pin is set to HIGH

	Example	int x=13; //connect 'x' to pin 13
		int p=7; //connect push button to pin 7
		int val=0; //variable to store the read value
		void setup()
		{
		pin MODE(x, OUTPUT); // sets 'x' as OUTPUT
		}
		void loop()
		{
		val=digital Read (p); //sets 'value' to 0
		digital Write (x,val); //sets 'x' to button value
		}

Analog i/o	**Methods**	**Usage**
	analog Read(pin)	Reads value from a specified analog pin works on pins 0–5. val=anolog Read (pin); // 'val' equal to pin
	analog Write (pin,value)	Writes an analog value using pulse width modulation (PWM) to a pin marked PWM works on pins 3,5,6,9,10.

	Example	int x=10; //connect 'x' to pin 13
		int p=0; //connect potentiometer to analog pin 7
		int val; //variable for reading
		void setup () { } //No setup is needed
		void loop()
		{
		val=analog Read (p); //sets 'value' to 0
		val/=4;
		analog Write (x,val); //outputs PWM signal to 'x'
		}

time	Methods	Usage
	deaay (ms)	Pauses for amount of time specified in milliseconds. deay(1000); //waits for one second
	millis()	Returns the number of milliseconds since Arduino is running. val=millis();//'val' will be equal to millis()

math	Methods	Usage
	min(x,y)	Calculates minimum of two numbers val=min (val,10); //sets 'val' to smaller than 10 or equal to 10 but never gets above 10.
	max(x,y)	val=max (val, 10); //sets 'val' to larger than 100 or 100.
random	**Methods**	**Usage**
	Random Seed (value)	Sets a value / seed as starting point for ranction.
	Random (min, max)	Allows to return numbers within the range specified by min and max values. val=random(100,200); //sets 'val' to random number between 100-200

	Example	intr number ; //variable to store random value
		int x=10;
		void setup()
		{
		randomseed (millis()); //set millis() as seed
		number -random(200); //random number from 0–200
		analog Write(x, number); //outputs PWM signal
		delay (500);
		}

Serial	**Methods**	**Usage**

	Serial. begin(rate)	Opens serial port and sets the baud rate for serial data transmission.
		void setup()
		{
		Serial begin(9600); //sets default rate to 9600 bps
		}

	Serial. println (data)	Prints data to the serial port Serial println (value); //sends the 'value'; //sends the ' value ' to serial monitor.

5.4.1) Difference between Analog, Digital and PWM Pins

- In analog pins, you have unlimited possible states between 0 and 1023.
- This allows you to read sensor values.
- Example: With a light sensor, if it is very dark , you will read 1023,if it is very bright yow will read as 0.
- If the brightness between dark and very bright you will read a value between 0 and 1023.
- In digital pins you have just two possible states, which is on and off.
- These can also be referred as High or Low,1 or 0 and 5v or 0V
- If the LED is on then its state is High or 1 or 5 V. If it is off have Low 0 0r 0 v.
- PWM pins are digital pins , so they output either 0 or 5v
- However these pins can output fake intermediate voltage values between 0 and 5 v because they can perform " Pulse Code Modulation".

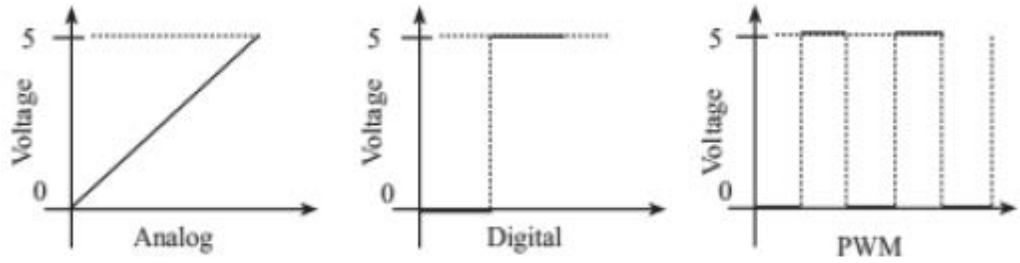


Figure 5.7 : Difference between Analog, Digital and PWM Pins.

5.5) Introduction to RaspberryPi

- The RaspberryPi is a series of credit card sized single-board computers developed in the United Kingdom by the Raspberrypi Foundation to promote the teaching of basic computer science in schools and developing countries.
- Several generations of RaspberryPi have been released.
- The first generation (RaspberryPi 1 model B) was released in February 2012, followed by a simple and inexpensive model A.
- In 2014 , the foundation released a board with an improved design in Raspberry 1 model B+.
- Improved A+ and B+ model were released a year later.
- RaspberryPi Zero with smaller size and limited input/output (I/O)and general purpose input/ output (GPIO) abilities was released in November 2015 for US $5.
- RaspberryPi 2 which added more RAM was released in February 2015.
- RaspberryPi 3 model B released in February 2016 in bundled with on-board Wi-Fi and Bluetooth.
- As of 2016 , Raspberrypi 3 model b is the newest mainline RaspberryPi. These boards are priced between US $ 5-35.

RaspberryPi	Model A+	Model B	Model B+	2, Model B	Model 3
Quick Summary	Cheapest,smallest single board computer	The original Raspberry Pi.	More USB and GPIIO than the B.Ideal choice for schools	most advanced Raspberry Pi.	Newest with wireless connectivity
Chip	Broadcom BCM 2835			Broadcom BCM2836	Broadcom BCM 2837
processor	ARMv6 single core			ARMv7 quad core	4×ARM Cortex-A53
Processor speed	700 MHz			900 MHz	1.2GHz
Voltage and power draw	600mA @ 5V			650mA @ 5V	
GPU	Dual core Videocore IV Multimedia Co-Processor				Broadcom Videocone IV
Size	65×56mm	85×56mm			
Memory	256 MB SDRAM @ 400 MHz	512 MB SDRAM @ 400 MHz		1 GB SDRAM @ 400 MHz	1 GB LPDDR2 (900 MHz)
Storage	Micro SD Card	SD Card	Micro SD Card	Micro SD Card	Micro SD Card
GPIO	40	26	40		
USB 2.0	1	2	4		
Ethernet	None	10/100mb Ethernet RJ45J ack			
Wireless	None				2.4GHz 802.11n wireless
Bluetooth	None				Bluetooth 4.1 Classic, Bluetooth Low Energy
Audio	Multi-Channel HD Audio over HDMI, Analog Stereo from 3.5mm Headphone Jack				

Operating Systems	Raspbian RaspBMC,Arch Linux,Rise OS,OpenELEC Pidora
Video Output	HDMI Composite RCA
Supported Resolutions	640×350 to 1920×1200, including 1080p,PAL & NTSC standards
Power Source	Micro USB

5.6) Exploring the RaspberryPi Learning Board

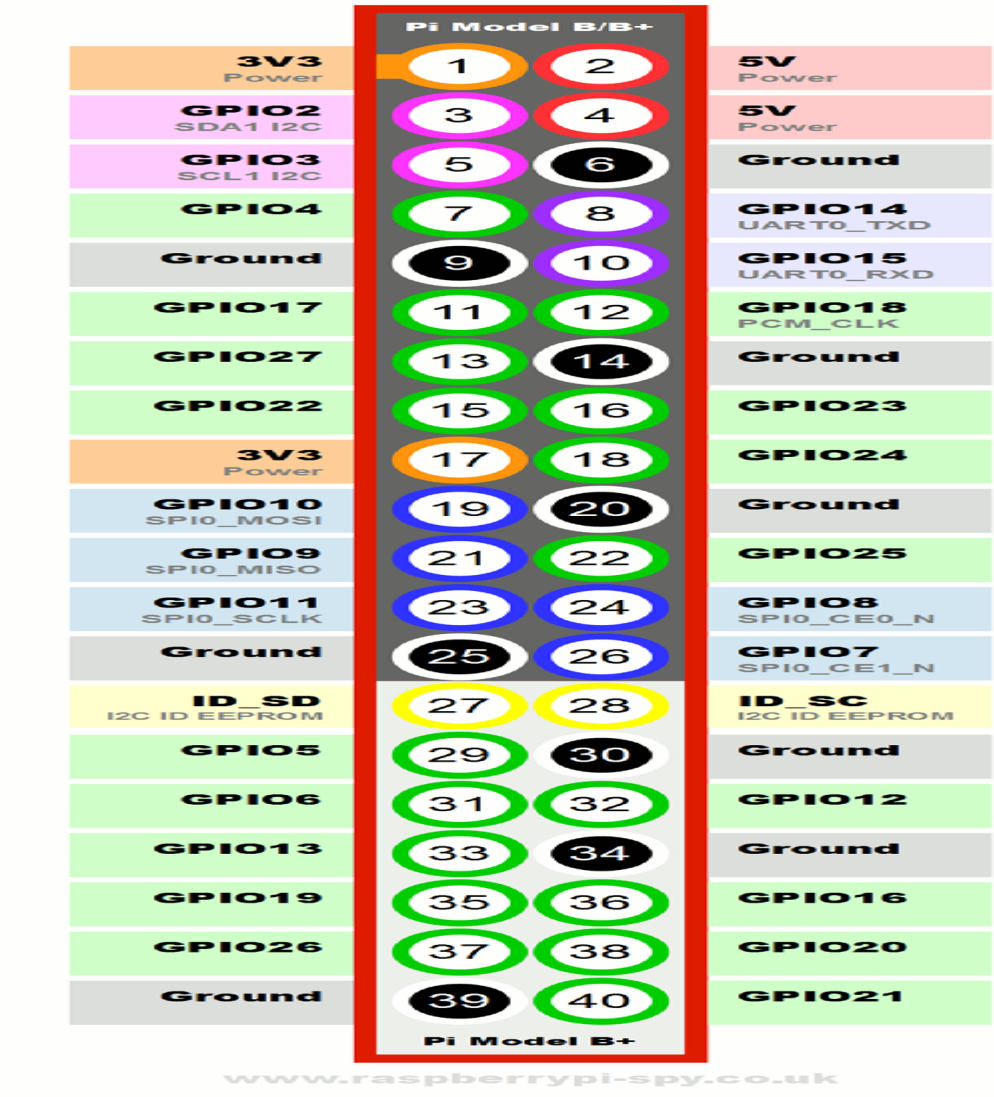


Figure 5.8: GPIO Pinout Diagram.

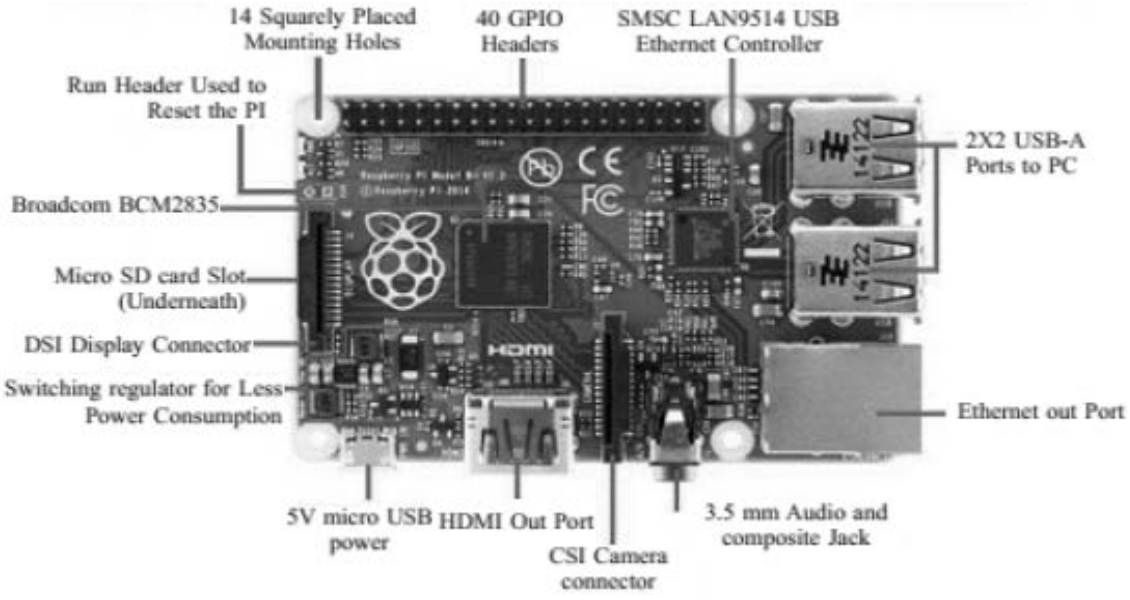


Figure 5.9: Raspberry Pi2 Model B and its GPIO.

- **Processor –** The Broadcom BCM2835 SoC(System on Chip) used in the first generation Raspberrypi is somewhat equivalent to the chip used in first generation smart phones, Which includes a 700MHz ARM1176JZF-S Processor, Video Core IV graphics processing unit(GPU) and RAM.
- This has a level 1 (L1) cache of 16 KB and a level 2 (L2) cache of 128 KB.
- The level 2 cache is used primarily by the GPU.
- The Raspberrypi2 uses a Broadcom BCM2836 SoC with a 900 MHz 32-bit quad-core ARM cortex A7 processor with 256KB shared L2 cache.
- The Raspberrypi2 uses a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM cortex A53 processor, with a 512 KB shared L2 cache.
- **Power Source**:- The recommended and easiest way to power the Raspberrypi is via the Micro USB port on the side of the unit.
- The recommended input voltage is 5V , and the recommended input current is 2A.
- **SD Card (Secure Digital Card**): The Raspberry Pi does not have any locally available storage accessible.
- The working framework is stacked on a SD card which is embedded on the SD card space on the Raspberry Pi.
- GPIO(General Purpose Input Output): General –purpose input/output (GPIO) is a non specific pins on a coordinated circuit to know is an input or output pin which can be controlled by the client at run time.
- GPIO capabilities may include

1. GPIO pins can be designed to be input or output.

2. Input values are meaning (normally high=1, low=0).

3. Yield values are writable/meaningful.

4. Input values can frequently be utilized as IRQs(interrupt request).

- **DSI Display x:** The Raspberrypi Connector S2 is a display serial interface(DSI) for connecting a liquid crystal display(LCD) panel using a 15 pin ribbon cable.
- **Audio Jack:** A standard 3.5 mm TRS connector is accessible on the Rpi for stereo sound yield.
- Any earphone or 3.5mm sound link can be associated straightforwardly.
- **Status LEDS:** There are 5 status LEDs on the Rpi that demonstate the status of different exercise

1. OK- SDCard Access.
2. POWER- 3.3 v Power
3. FDX- Full Duplex LAN
4. LNK- Link/ Activity(LAN) (Model B)
5. 10M/100-10/100M bit (LAN) (Model B)

❖ **Ethernet Port :**is accessible on Model B and B+.

- It can be associated with a system or web utilizing a standard LAN link on the Ethernet port.

❖ **CSI connector (CSI) –** Camera Serial interface is a serial interface outlined (define) by MIPI (Mobile Industry Processor Interface) organization together went for interfacing cameras with a portable processor.

❖ **HDMI-** High Definition Multimedia Interface to give both video and sound yield.

5.6.1) Description of system on chip(SoC)

- SoC is an integrated circuit(IC) that coordinated all parts of a PC or other electronic framework into a solitary chip.
- SoC comprises of:

1) A microprocessor chip or DSP core.

2) Memory pieces including (ROM,RAM,EEPROM)

3) Timing signal –oscillators

4) Peripherals include counter-clocks, ongoing clocks

5) Outer interfaces example: USB, Ethernet

6) Simple interfaces includes ADCs (Analog and Digital Converter) and DACS(Digital and Analog Converter)

7) Voltage Controllers and power administration circuits.

Accessories

❖ **Camera:** On 14 May 2013 , the establishment and the merchants RS Components and Premier Farnell/ Element 14 propelled the Raspberry pi camera board with a firmware redesign to bolster it.

❖ **Gertboard**- A Raspberry Pi Foundation authorized gadget intended for instructive purpose, and grows the Raspberry Pi's GPIO pins to permit interface with of LEDs , switches, sensors and different gadges.

5.7) Raspberry Pi interfaces.

❖ **Serial :-** The serial interface on Raspberry Pi has receive(rx) and transmit(Tx) pins for communication with serial peripherals.

❖ **SPI:-** Serial Peripheral interfaces(SPI) is a synchronous serial data protocol used for communication with one or more peripheral devices.

- MISO (Master In Slave Out): Master line for sending data to the peripherals.
- MOSI(Master out Slave In): Slave line for sending data to the master.
- SCK(Serial Clock): Clock generated by Master to synchronize data transmission.
- CEO(Chip Enable 0): To enable or disable device
- CEO(Chip Enable 1): To enable or disable device

❖ **I2C:-** The I2C interface pins on Raspberry Pi allow you to connect hardware modules.

5.8)RaspberryPI Operating System.

- Various operating system can be installed on Raspberrypi through SD cards.

5.8.1) Operating Systems(not Linux based)

✓ RISC OS Pi (a special cut down version RISC OS Pico, for 16MB cards and larger for all models of Pi 1 and 2, has also been made available.

✓ FreeBSD

✓ NetBSD

✓ Plan 9 from Bell Labs and Inferno

✓ Windows 10 IoT core- a no cost edition of Windows 10 offered by Microsoft that runs natively on the RaspberryPi 2.

✓ xv6-is a modern reimplementation of sixth edition Unix OS for teaching purposes, it is ported to RaspberryPi from MIT Xv6, which can boot NOOBs.

✓ Haiku-is an open source BeOS clone that has can be compile for the RaspberryPi and several other ARM boards

5.8.2) Operating Systems(Linux based)

- ✓ Xbian-using Kodi open source digital media center
- ✓ openSUSE
- ✓ Raspberry Pi Fedora remix
- ✓ Pidora, another Fedora Remix optimized for the RaspberryPi
- ✓ Gentoo Linux
- ✓ Diet Pi
- ✓ CentOS\OpenWrt
- ✓ Kali Linux
- ✓ Ark OS
- ✓ Kano OS
- ✓ Nard SDK

5.8.3) Media center Operating systems

- ✓ OSMC
- ✓ OpenELEC
- ✓ LibreELEC
- ✓ Xbian
- ✓ Rasplex

5.8.4) Audio Operating systems

- ✓ Volumio
- ✓ Pimusicbox
- ✓ Runeaudio
- ✓ moOdeaudio

5.8.5) Recalbox

- ✓ Happi Game Center
- ✓ Lakka
- ✓ ChameleonPi
- ✓ Piplay

5.9) Operating System Setup in RaspberryPI

- Preinstalled NOOBS operating system is already available in many authorized as well as independent seller, there are many other operating system for Raspberrypi in the market like NOOBS, Raspbian and third party operating systems are also available like UBUNTU MATE, OSMC,RISC OS etc.
- To setup an operating system we need a SD card with minimum capacity of 8GB.

5.9.1) Formatting SD card:

Format the SD card before copying NOOBS onto it. To do this

1. Download SD formatter 4.0 from SD Association website for either Windows or Mac.
2. Follow the instructions to install the software.
3. Insert the SD card into the computer or laptops SD card reader and make a note of the drive letter allocated to it
4. In SD formatter, select the drive letter the SD card is and format it.

5.9.2) OS installation

Follow the steps to install operating system in the SD card.

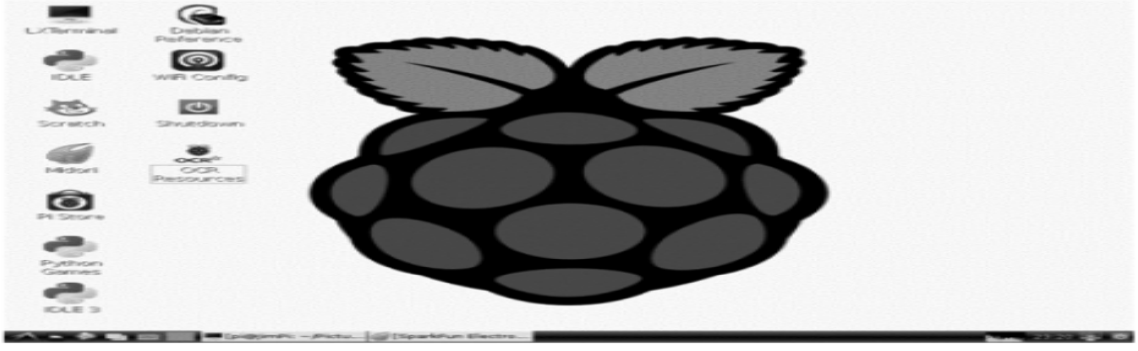
1. Go to raspberry pi foundation website and click on DOWNLOAD section.
2. Click on NOOBS, then click on the "Download ZIP" button under NOOBS(offline and network install) and select a folder to save this ZIP file
3. Extract all the files from ZIP.
4. Once SD card has been formatted, drag all the files in the extracted NOOBS folder and drop them onto the SD card drive.
5. The necessary files will then be transferred to the SD card.
6. When this process has finished, safely remove the SD card and insert it into the Raspberry Pi.

5.9.3) First Boot

1. Plug in the keyboard, mouse, and monitor cables.
2. Now plug the USB cable into the Raspberrypi.
3. Now Raspberrypi will boot, and a window will appear with a list of different operating systems that we can install. We recommend using Raspbian- tick the box next to Raspbian and clicking on install.
4. Raspbian will then run through its installation process. Note that this can take a while.
5. When the install process has completed, the Raspberrypi configuration menu (raspi-config) will load. Here we should set the time and date for our region, enable a Raspberrypi camera board, or even create users.

5.10) Login information

- The default login for Raspbian is username "pi" with the password "raspberry".
- To load the graphical user interface, type "Startx" and press Enter.



5.10) **RaspberryPI Commands**

General commands for RaspberryPi:

a. **raspi-config:** Configuration settings menu
b. **clear:** clears data from terminal.
c. **date:** current date.
d. **reboot:** Reboot immediately
e. **shutdown –**h now: Shutdown system immediately
f. **nano example.txt:** Opens the example.txt in the text editor nano.
g. **poweroff:** To shutdown immediately.
h. **shutdown –**h 01:22: To shutdown at 1:22 AM.
i. **apt-get update:** Synchronizes the list of packages on our system to the list in the repositories. Use this before installing new packages to make sure we are installing the latest version
j. **apt-get upgrade:** Upgrades all of the software packages we have installed.
k. **startx:** Opens the GUI

Directory and File commands:

a. **mkdir new_directory**: Creates a new directory named new_directory.
b. **mv new_folder:** Moves the file or directory named "new_folder" to a specified location
c. **rm new_file.txt:** Deleted the file new_file.txt.
d. **rmdir new_directory:** Deletes the directory "new_directory" only if it is empty.
e. **touch new_file.txt:** creates a new, empty file named new_file.txt in the current directory.
f. **cat new_file.txt:** Displays the contents of the file new_file.txt
g. **cd /xyz/abc:** Changes the current directory to the /xyz/abc directory.
h. **ls –l:** lists files in the directory.

Networking and Internet Commands:

a. **iwconfig:** Tp check which wireless adapter is currently active.
b. **ifconfig**: wireless connection ststus.
c. **ping:** tests connectivity between two devices connected on a network.
d. **wget http://www.website.com/new_file.txt**: Downloads the file new_file.txt from the web and saves it to the current directory.
e. **nmap:** Scans the network and lists connected devices, protocol, port number and other information.
f. **iwlist wlan0 scan:** list of currently available wireless networks.

System information commands:

a. **cat /proc/meminfo:** shows details about memory
b. **cat /proc/version:** shows which version of rsapberrypi we are using.
c. **df –h:** shows information about available disk space.
d. **df /:** shows how much free disk space is available.
e. **free:** shows how much free memory is available.
f. **hostname –I:** shows th ip address of the raspberrypi.
g. **lsusb;** lists the usb hardware connected to raspberrypi.
h. **vcgencmd measure_temp:** shows the temperature of the cpu.
i. **vcgencmd get_mem arm && vcgencmd get_mem gpu:** shows the memory split between the cpu and gpu.

5.11) Programming RaspberryPI with python

Program	Code
Print hello world	print("hello world")
program to add two numbers **code**	a=1.2 b=5.3 sum=float(a)+float(b) print("the sum of {0} and {1} is {2}".format(a,b,sum))
program to roll a dice	import random min=1 max=6 roll_again="yes"
	while roll_again=="yes" or roll_again=="y" print("rolling the dices") print("the values are") print(random.randomint(min,max) print(random.randint(min,max)
program to find the ip address of raspberrypi	import urllib import re print("we will try to open this url, in order to get ip address") url=http://checkip.dyndns.org print(url)
program to generate password	import string from random import* characters=string.ascii_letters+string.punctuation+string.digits password="".join(choice(charcters) for x in range(randint(8,16))) print(password)
program to print fibonacci series	a,b=0,1 while b<200: print(b) a,b=b,a+b

program to check for armstrong number	num=int(input("enter a number:"))
	initial_sum=0
	temp=num
	while temp>0:
	digit=temp%10
	initial_sum+=digit**3
	temp//=10
	if num==initial_sum:
	print(num,"is an armstrong number")
	else:
	print(num,"is not an armstrong number")

program to display calendar of given month of the year	import calendar yy=2017 mm=11 print(calendar.month(yy,mm))

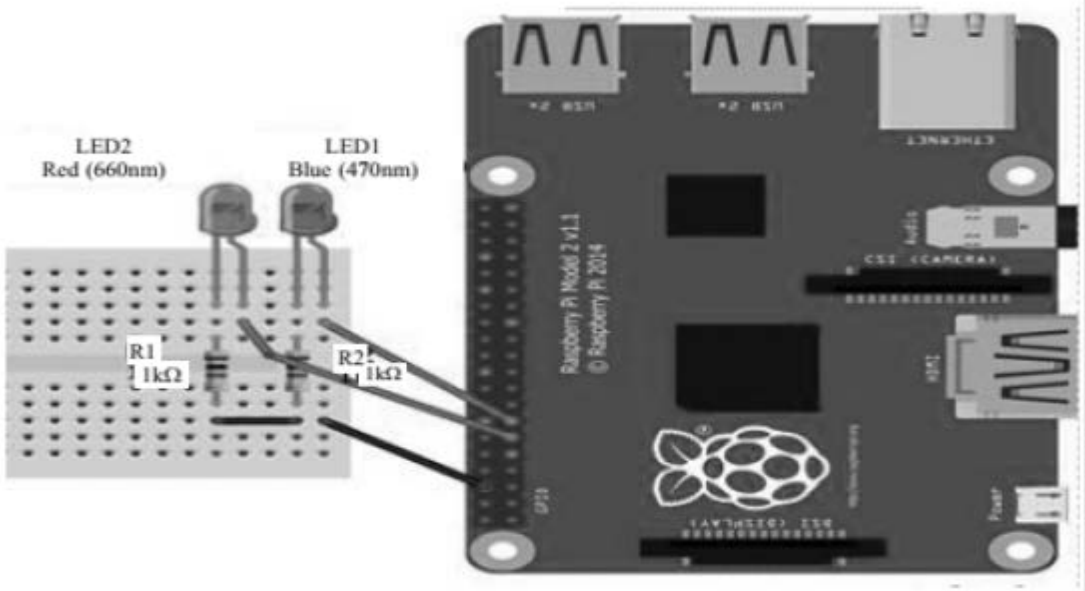
Program #1	Printing to a terminal
Components required	Raspberry Pi + SD card, Monitor + HDMI Cable, Keyboard & Mouse and Power supply
Description	Now let us start with a basic example of printing a message "Hello World" using Python Programming. Box shows how to print a greeting message to the console.
Key points	1. Find your customized Raspberry Pi. 2. Mount the SD card. 3. Plug in the HDMI cable into the Pi and the monitor. 4. Plug in the keyboard into the USB ports 5. Plug in the mouse into the USB ports 6. Plug in the power cable 7. Type in user name "pi" 8. Type in password "raspberry" 9. Double click on "Terminal" 10. This will load the "terminal" 11. Type the follow commands ✓ Change the directory by the command $ cd Desktop ✓ Create a new directory $ mkdir python_code ✓ Change the directory to python_code $ cd python_code ✓ create new file helloworld.py ✓ Now enter the code given in the box below ✓ Run the python code "sudo python helloworld.py ✓ You will see it print "Hello World!" to the screen

Code	**File:Helloworld.py**
	#Access the python working environment
	#!/usr/bin/python
	#Print a message Hello world on to the terminal
	print("Hello World!")

Output	A message "Hello world" will prints on the console.

Program #2	Blinking an LED
Components required	Raspberry Pi + SD card, Monitor + HDMI Cable, Keyboard & Mouse and Power supply, 1 Red LED and Blue LED, 2 1K resistors.
Description	Now let us look at an example of controlling the state of LEDs from ON to OFF state or vice versa from Raspberry Pi. Figure shows the schematic diagram of connecting an LEDs to Raspberry Pi. Box shows how to turn the LED on/off from by running the code given. In this example the LEDs are connected to GPIO pin 17 and 27 respectively which is initialized as output pin. The state of the LED is toggled by the executing the two programs given in the Box. Box
	having the code changes the state of the LED twice whereas Box having the code runs an infinite loop to flicker LEDs from ON/OFF state every second. Run the code given below and observe at the desired output.

Circuit Diagram



Key points	1. Create file "blink.py" 2. Create file "blink_ever.py" 3. Enter the above code 4. Run the python file "sudo python blink.py" << Watch the LEDs blink 2 times
	5. Run the python file "sudo python blink_ever.py" << Watch the LEDs blink forever
Code	**File: blink.py**

```
#Access the python working environment
#!/usr/bin/python
#import the time module so as to switch LEDs on/off with the time elapsed
#import the RPI.GPIO library
import RPi.GPIO as GPIO
#use one of the two numbering system either BOARD numbers/BCM
# Refer to the channel numbers on the Broadcom SOC.
GPIO.setmode(GPIO.BCM)
#Configure Pin 17 as an OUTPUT
GPIO.setup(17,GPIO.OUT)
#Configure Pin 27 as an OUTPUT
GPIO.setup(27,GPIO.OUT)
#Turn up LEDs on pin 17
GPIO.output(17,GPIO.HIGH)
#Turn up LEDs on pin 27
GPIO.output(27,GPIO.HIGH)`
#wait for 1 second
time.sleep(1)
#Turn up LEDs off on pin 17
GPIO.output(17,GPIO.LOW)
#Turn up LEDs off on pin 27
GPIO.output(27,GPIO.LOW)
#wait for 1 second
time.sleep(1)
```

File:blink_ever.py

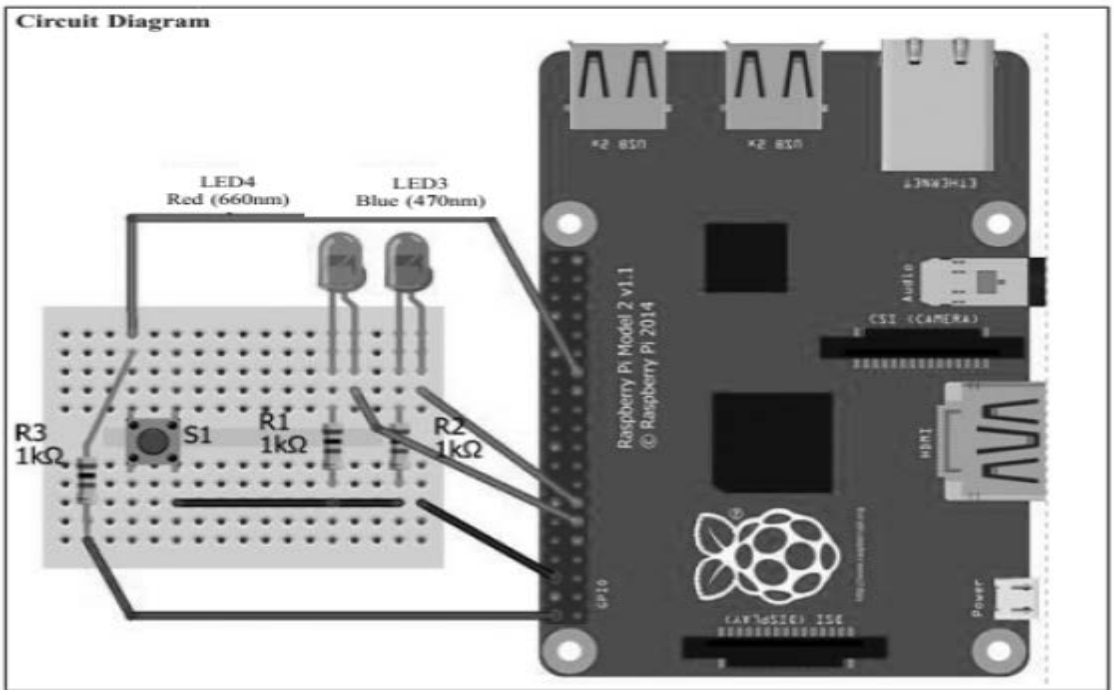
```
#Access the python working environment
#!/usr/bin/python
#import the time module so as to switch LEDs on/off with the time elapsed
import time
#import the RPI.GPIO library
```

```
import RPi.GPIO as GPIO
#use one of the two numbering system either BOARD numbers/BCM
# Refer to the channel numbers on the Broadcom SOC.
GPIO.setmode(GPIO.BCM)
#Configure pin 17 and 27 to be an OUTPUT pins
GPIO.setup(17,GPIO.OUT)
GPIO.setup(27,GPIO.OUT)
#Use while construct which runs infinite number of times there by blinking
        LEDs forever
while 1:
        #Turn up LEDs on
      GPIO.output(17,GPIO.HIGH)
       GPIO.output(27,GPIO.HIGH)
      time.sleep(1)
       #Turn up LEDs off
      GPIO.output(17,GPIO.LOW)
      GPIO.output(27,GPIO.LOW)
      time.sleep(1)
```

Output	LEDs turns on/off twice when blink.py file is executed and LEDs keeps changing their state forever when blink_forever.py file is executed.

Program #3	**Push button for physical input**
Components required	Raspberry Pi + SD card, Monitor + HDMI Cable, Keyboard & Mouse and Power supply, 1 Red LED and Blue LED, 2 1K resistors, push button and jumper wires.
Description	Now let us look at an example involving LEDs and a switch that is used to control LED. Figure shows the schematic diagram of connecting an LEDs and a switch to Raspberry Pi. Box shows a python program for controlling an LED with a switch.In the infinite while loop the value of pin 10 is checked and the state of the LED is toggled if the switch is pressed. This example shows how to get input from GPIO pins and process the state of the LEDS. Run the code given below and observe at the desired output.

Circuit Diagram



Key points	1. Create file "button.py" 2. Enter the above code 3. To run the python code "sudo python button.py"
Code	File: button.py #Access the python working environment #!/usr/bin/python #Import os module to enable interrupts from a push button import os #import the time module so as to know the time the user as given the input from a push button

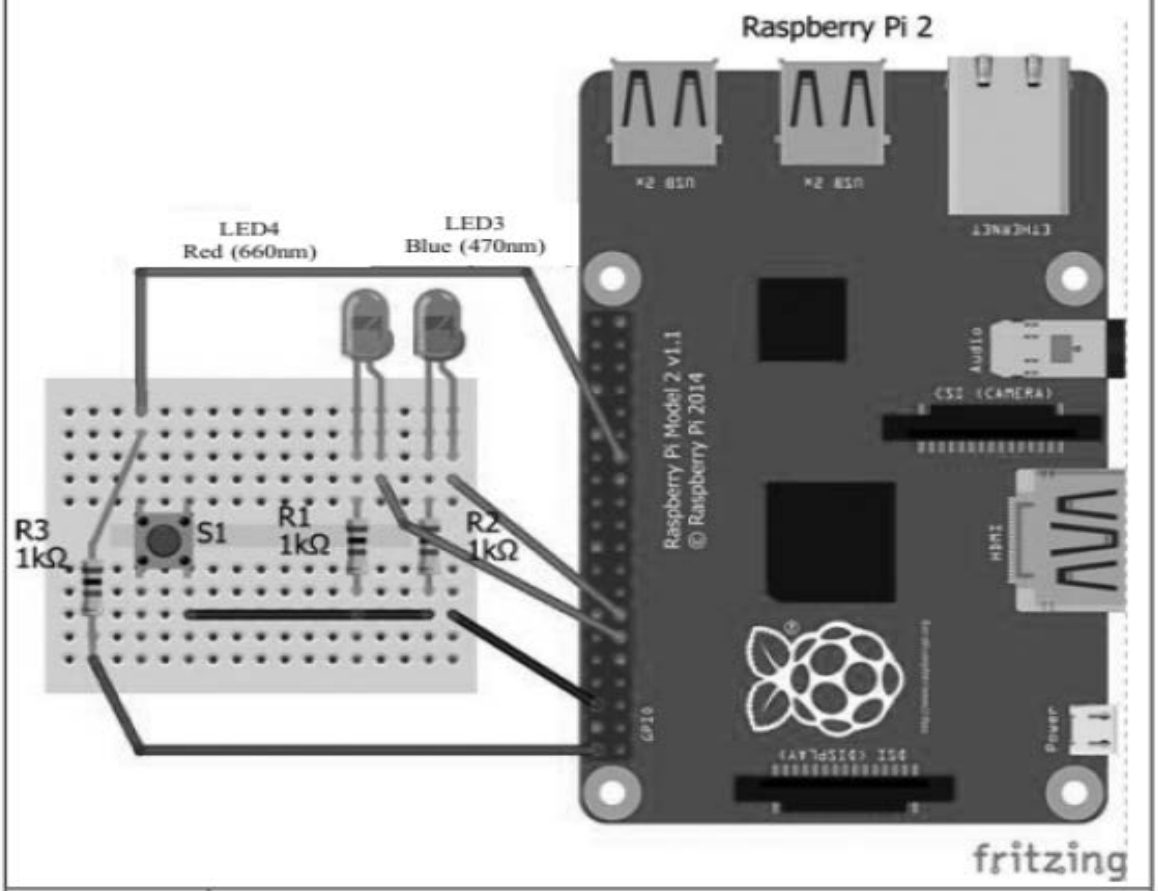
```
import time
#import the RPI.GPIO library
import RPi.GPIO as GPIO
#use one of the two numbering system either BOARD numbers/BCM
# Refer to the channel numbers on the Broadcom SOC.
GPIO.setmode(GPIO.BCM)
#configure pin 10 to be INPUT which reads the status of a switch button
GPIO.setup(10, GPIO.IN)
#print a message on to the terminal
print(" Button + GPIO ")
#Read the status of a button from GPIO pin 10
print GPIO.input(10)
#Run a infinite loop on the status of the button read
while True:
  if ( GPIO.input(10) == True ):
      print("Button Pressed")
     #Print the time when the input was given from the push button
      os.system('date')
     #Read the status of a button from GPIO pin 10
      print GPIO.input(10)
     #wait for 5 seconds
      time.sleep(5)
```

```
else:
    #clear the system variables
    os.system('clear')
    #prompt the user to give an input
    print ("Waiting for you to press a button")
time.sleep(1)
```

Output	Press the Push button switch to Turn ON/OFF the LED

Program #4	**Interact with the user**
Components required	Raspberry Pi + SD card, Monitor + HDMI Cable, Keyboard & Mouse and Power supply, 1 Red LED and Blue LED, 2 1K resistors, push button and jumper wires.
Description	Now let us look at an example involving LEDs and a switch that is used to control LED depending on what a user choose. Figure shows the schematic diagram of connecting an LEDs and a switch to Raspberry Pi. Box shows a python program for controlling an LED by reading two input values, one for which LED would user like to blink(option 1-for Red, 2- for Blue) and one more parameter to set the maximum number of times the LED should be flickered. This example shows how to get input from a user and process the state of the LEDS. Run the code given below and observe at the desired output.

Circuit Diagram



Key points	1. Create file "user_input.py" 2. Enter the above code

	3. Run the python file by "sudo python user_input.py" << Run through the questions and make an LED blink.

Code	**File: user_input.py**
	#Access the python working environment
	#!/usr/bin/python
	#Import os module to enable interrupts from a push button
	import os
	#import the time module so as to switch LEDs on/off with the time elapsed
	import time
	#import the RPI.GPIO library
	import RPi.GPIO as GPIO
	#use one of the two numbering system either BOARD numbers/BCM
	# Refer to the channel numbers on the Broadcom SOC
	GPIO.setmode(GPIO.BCM)
	#configure pin 17 to be OUTPUT pin
	GPIO.setup(17,GPIO.OUT)
	#configure pin 27 to be OUTPUT pin
	GPIO.setup(27,GPIO.OUT)
	
	#Initialize variables for user input
	led_ch = 0
	counter = 0
	
	#clear the python interpreter console
	os.system('clear')
	
	print "Which LED would you like to blink"
	#Accept 1 for Red
	print "1: Red?"
	#Accept 2 for Blue
	print "2: Blue?"
	
	led_ch = input("Choose your option: ")
	
	if led_ch == 1:

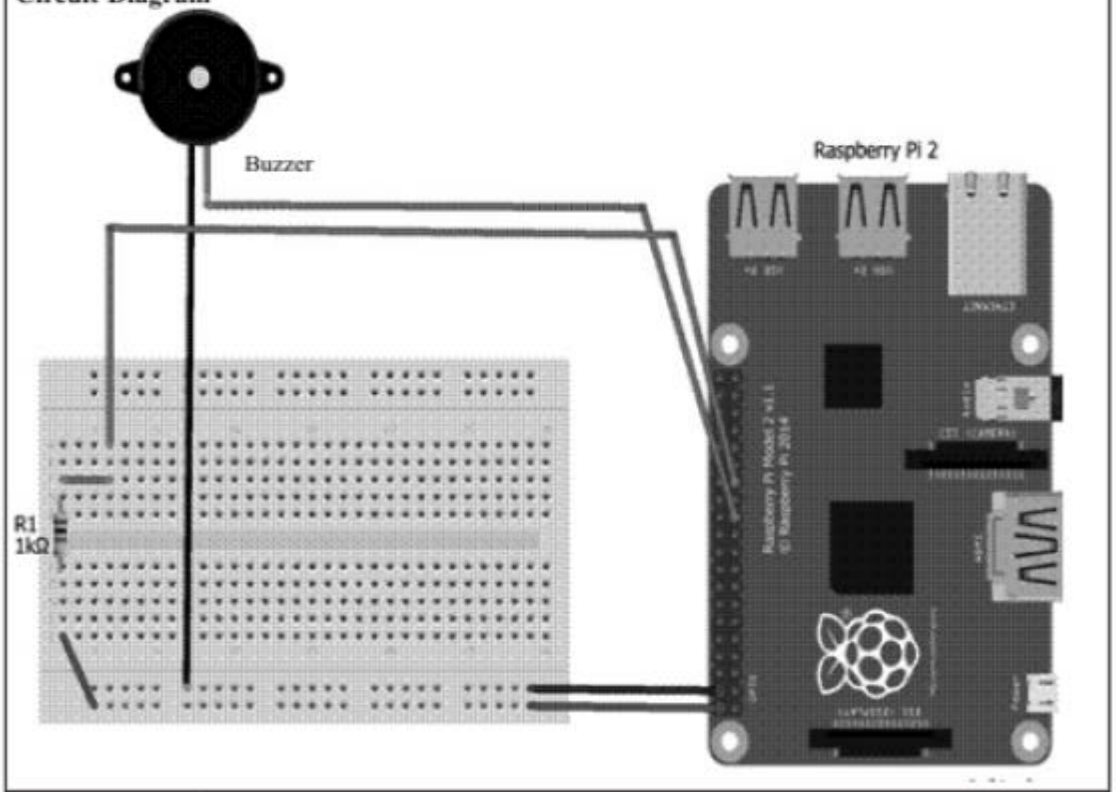
	#clear the python interpreter console
	os.system('clear')
	print "You picked the Red LED"
	counter = input("How many times would you like it to blink?: ")
	while counter > 0:
	#on LED on Pin 27
	GPIO.output(27,GPIO.HIGH)
	time.sleep(1)
	#off LED on pin 27
	GPIO.output(27,GPIO.LOW)
	time.sleep(1)
	#Record the number of counts on LED
	counter = counter – 1

```
if led_ch== 2:
        #clear the python interpreter console
        os.system('clear')
        print "You picked the Red LED"
        counter = input("How many times would you like it to blink?: ")
        while counter > 0:
                #on LED Pin 27
                GPIO.output(17,GPIO.HIGH)
                time.sleep(1)
                #off LED on pin 27
                GPIO.output(17,GPIO.LOW)
                time.sleep(1)
                #Record the number of counts on LED
                count = count – 1
```

Output	LED gets flickered on the inputs given by the user.

Program #5	**Buzzer**
Components required	Raspberry Pi + SD card, Monitor + HDMI Cable, Keyboard & Mouse and Power supply, 1 Red LED and Blue LED, 2 1K resistors, push button and jumper wires, Breadboard, buzzer.
Description	Now let us look at an example involving a Piezo buzzer and a switch that is used to beep a number of times the user choose. Figure shows the schematic diagram of connecting an piezo buzzer to pin 22 and a switch to Raspberry Pi. Box shows a python program for controlling an piezo buzzer by reading an input value which runs over a loop to beep number of times the user has choose. Run the code given below and observe at the desired output.

Circuit Diagram



Key points	1. Create file "buzzer.py" 2. Enter the code above code 3. To run python code "sudo python buzzer.py"<<listen to it beep

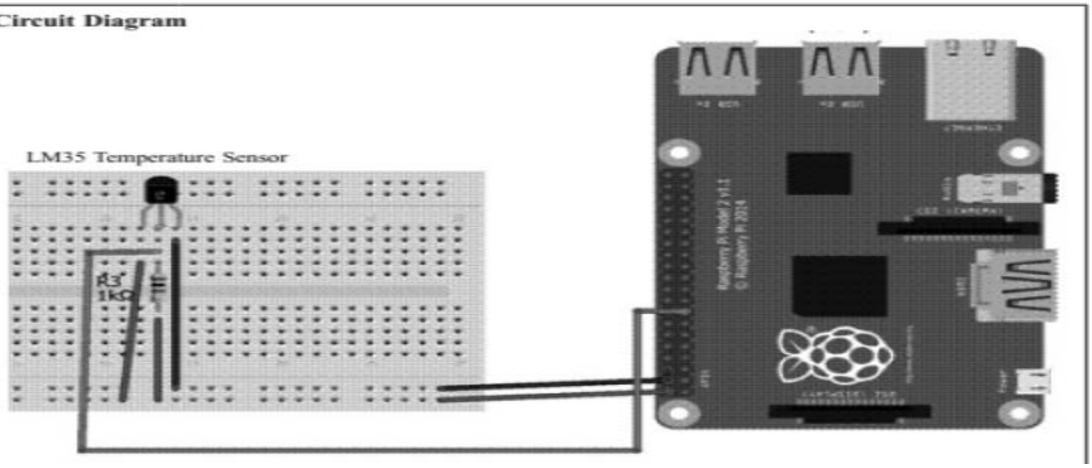
Code	**File: buzzer.py**
	#!/usr/bin/python
	import os
	import time
	import RPi.GPIO as GPIO
	GPIO.setmode(GPIO.BCM)
	GPIO.setwarnings(False)
	GPIO.setup(22,GPIO.OUT)
	loop_counter = 0
	def morsecode ():
	#Dot Dot Dot
	GPIO.output(22,GPIO.HIGH)
	time.sleep(.1)
	GPIO.output(22,GPIO.LOW)
	time.sleep(.1)
	GPIO.output(22,GPIO.HIGH)
	time.sleep(.1)
	GPIO.output(22,GPIO.LOW)
	time.sleep(.1)
	GPIO.output(22,GPIO.HIGH)
	time.sleep(.1)
	#Dash Dash Dah
	GPIO.output(22,GPIO.LOW)
	time.sleep(.2)
	GPIO.output(22,GPIO.HIGH)
	time.sleep(.2)
	GPIO.output(22,GPIO.LOW)
	time.sleep(.2)
	GPIO.output(22,GPIO.HIGH)
	time.sleep(.2)
	GPIO.output(22,GPIO.LOW)
	time.sleep(.2)
	GPIO.output(22,GPIO.HIGH)
	time.sleep(.2)
	GPIO.output(22,GPIO.LOW)
	time.sleep(.2)

```
        #Dot Dot Dot
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.1)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.1)
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.1)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.1)
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.1)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.7)
os.system('clear')
print "Morse Code"
loop_count = input("How many times would you like SOS to loop?: ")
while loop_count > 0:
        loop_counter = loop_counter - 1
        morsecode ()
```

Output	listen to beep of piezo buzzer

Program #6	**Temperature sensor**
Components required	Raspberry Pi + SD card, Monitor + HDMI Cable, Keyboard & Mouse and Power supply, 1 Red LED and Blue LED, 2 1K resistors, push button and jumper wires, Breadboard, buzzer.1 LM35 temperature sensor.
Description	Now let us look at an example involving an DS18B22 temperature sensor which reads out a temperature and records on to a terminal. Figure shows the schema -tic diagram of connecting an DS18B22 temperature sensor to Raspberry Pi board. Box shows a python program which records the temperature read by LM35 temperature sensor.This example shows how to get an analog input from GPIO pins and process the input. An infinite loop runs over the sensor which records a temperature every second Compile the code given below and upload it to Arduino UNO Board to observe at the desired output.

Circuit Diagram

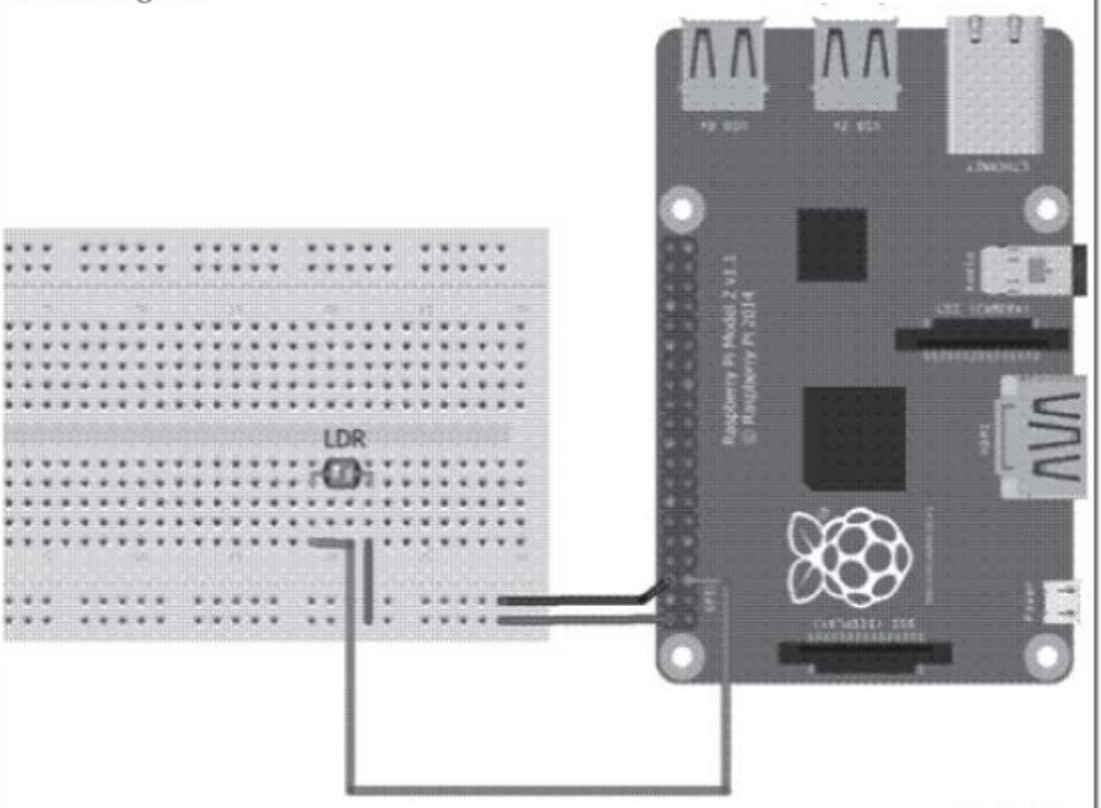


Code	import os
	import glob
	import time
	#initialize the device
	os.system('modprobe w1-gpio')
	os.system('modprobe w1-therm')
	
	base_dir = '/sys/bus/w1/devices/'
	device_folder = glob.glob(base_dir + '28*')[0]
	device_file = device_folder + '/w1_slave'
	
	def read_temp_raw():
	f = open(device_file, 'r')
	lines = f.readlines()
	f.close()
	return lines
	
	def read_temp():
	lines = read_temp_raw()
	while lines[0].strip()[-3:] != 'YES':
	time.sleep(0.2)
	lines = read_temp_raw()
	equals_pos = lines[1].find('t=')
	if equals_pos != -1:
	temp_string = lines[1][equals_pos+2:]
	temp_c = float(temp_string) / 1000.0
	temp_f = temp_c * 9.0 / 5.0 + 32.0
	return temp_c, temp_f
	
	while True:
	print(read_temp())
	time.sleep(1)

Output	Current Room Temperature is recorded.

Program #7	Light sensor
Components required	Raspberry Pi + SD card, Monitor + HDMI Cable, Keyboard & Mouse and Power supply, 1 Red LED and Blue LED, 2 1K resistors, push button and jumper wires, Breadboard, buzzer.LM35 temperature sensor, LDR Light Dependent resistor
Description	Now let us look at an example involving an LDR sensor which reads the intensity of light and records it an text file. Figure shows the schematic diagram of connecting an LDR sensor to Raspberry Pi board. Box shows a python program which records the intensity of light to a text file. This example shows how to get an analog input from GPIO pins and process the input. An infinite loop runs over the sensor which records intensity of light with date and time stamps every second Compile the code given below and upload it to Arduino UNO Board to observe at the desired output.

Circuit Diagram



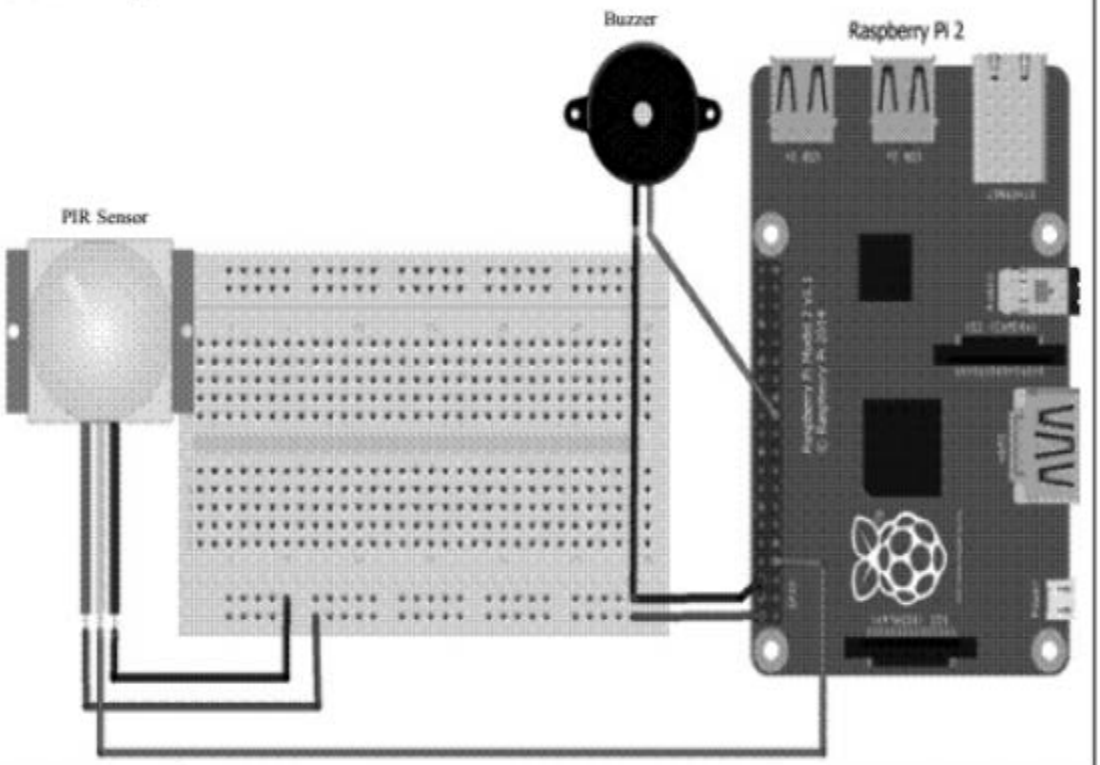
Key points	1. Create file "ldr.py" then "touch foo.txt" 2. To run the python code "sudo python ldr.py" << See what the light levels in the room are. 3. The check the file "more foo.txt" you can see your results.
Code	**File: ldr.py** #!/usr/bin/env python import os import datetime import time import RPi.GPIO as GPIO

```
DEBUGER = 1
GPIO.setmode(GPIO.BCM)
def RCtimer (RCpins):
        readings = 0
       GPIO.setup(RCpins, GPIO.OUT)
        GPIO.output(RCpins, GPIO.LOW)
       time.sleep(.1)

       GPIO.setup(RCpins, GPIO.IN)
        # iterates 1 miliseconds over one cycle
       while (GPIO.input(RCpins) == GPIO.LOW):
              readings += 1
       return readings

while True:
                    GetDateTime = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")\
                    LDRReading = RCtimes(3)
                    print RCtimes(3)

                    # Open a file
                    fo = open("/home/pi/10x10/foo.txt", "wb")
                    fo.write (GetDateTime)
                    LDRReading = str(LDRReading)
                    fo.write ("\n")
                    fo.write (LDRReading)

                    # Close opend file
                    fo.close()
                    time.sleep(1)
```

Output	Intensity of the light in the room is recorded on to a terminal as well as to text file.

Program #8	**Passive Inferred Sensor**
Components required	Raspberry Pi + SD card, Monitor + HDMI Cable, Keyboard & Mouse and Power supply, 1 Red LED and Blue LED, 2 1K resistors, push button and jumper wires, Breadboard, buzzer.LM35 temperature sensor, LDR Light Dependent resistor, PIR(Passive Infrared sensor) sensor.
Description	Now let us look at an example involving an PIR sensor which detects the motion of an object. Figure shows the schematic diagram of connecting an PIR sensor to Raspberry Pi board. Box shows a python program which the motion of an object and triggers a message as "Motion detected".This example shows how to get an analog input from GPIO pins and process the input. An infinite loop runs over the PIR sensor which waits for any of the movements across its boundary. Compile the code given below and upload it to Arduino UNO Board to observe at the desired output.

Circuit Diagram



Key points	1. Create file "touch python pir.py" 2. To run the python code "sudo python pir.py" << Move in front of the PIR to activate it.
Code	**File: PIR.py** #!/usr/bin/python import RPi.GPIO as GPIO import time

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(27,GPIO.OUT)
GPIO_PIR_sensor = 7

print "PIR Module Test (CTRL-C to exit)"
# configure pin to be input
GPIO.setup(GPIO_PIR_sensor,GPIO.IN)

CurrentState  = 0
PreviousState = 0
```

```
try:
 print "Waiting for PIR to settle ..."
  # iterate till PIR  outputs the value 0
 while GPIO.input(GPIO_PIR_sensor)==1:
    CurrentState  = 0
 print "  Ready"
  # Iterate until user types CTRL-C
 while True :
    # status of the PIR to be read
   CurrentState = GPIO.input(GPIO_PIR_sensor)
    if CurrentState==1 and PreviousState==0:
     # Trigger action on PIR
      print "  Motion detected!"
     # Previous status of the PIR to be recorded
      GPIO.output(27,GPIO.HIGH)
     time.sleep(1)
      GPIO.output(27,GPIO.LOW)
     PreviousState=1
    elif CurrentState==0 and PreviousState==1:
     # check if PIR has arrived to the ready state
      print "  Ready"
      PreviousState=0
    # stop for 10 milliseconds
    time.sleep(0.01)
 except KeyboardInterrupt:
  print "  Quit"
   # GPIO settings to be reset
   GPIO.cleanup()
```

Output	Move in front of the PIR to activate it and sensor generates a message..

5.12) Digital Temperature Sensors Vs. Analog Temperature Sensors

- Analog temp. sensors contain thermistors and temperature value is obtained from resistance value (due to change in volt)
- Digital temperature sensors are typically silicon based integrated circuits
- Unlike analog temperature sensors, calculations are performed by the sensor, and the output is an actual temperature value

5.13) Raspberry Pi and DS18B20 Temperature Sensor

About the DS18B20

- The DS18B20 also has an alarm function that can be configured to output a signal when the temperature crosses a high or low threshold that's set by the user

- A 64 bit ROM stores the device's unique serial code. This 64 bit address allows a microcontroller to receive temperature data from many sensors with identity.
- The DS18B20 temperature sensor is perfect for projects like weather stations and home automation systems
- The size is same as a transistor and use only one wire for the data signal
- Extremely accurate and take measurements quickly

DS18B20 : Technical Specifications

- -55°C to 125°C -: temperature range
- 3.0V to 5.0V :- operating voltage
- 0.5°C (9 bit); 0.25°C (10 bit); 0.125°C (11 bit); 0.0625°C (12 bit) resolution
- 64 bit unique address
- One-Wire communication protocol

DS12B20 Pin Diagram

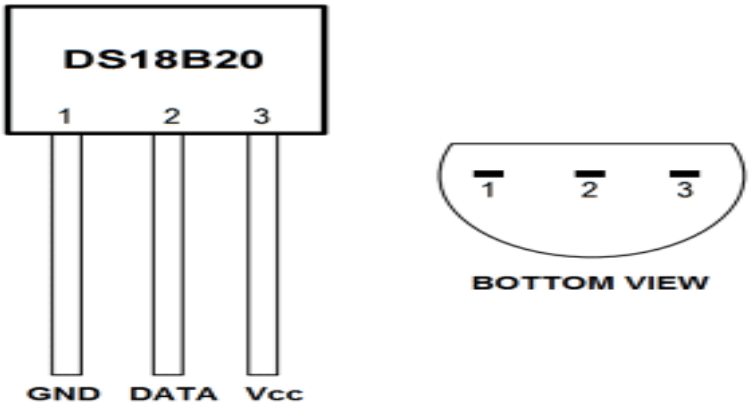


Figure 5.10 DS12B20 Pin Diagram

5.13.1) Configuring Raspberry Pi

- sudo nano /boot/config.txt
- Add this → "dtoverlay=w1-gpio" at the end of file if doesn't exist
- sudo reboot
- sudo modprobe w1-gpio

- sudo modprobe w1-therm
- cd /sys/bus/w1/devices
- ls
- cd 28-xxxx
- cat w1_slave
- a3 01 4b 46 7f ff 0e 10 d8 : crc=d8 YES
- a3 01 4b 46 7f ff 0e 10 d8 t=32768

5.14) Pi Via SSH (Secure Shell)

- You can access the command line of a Raspberry Pi remotely from another computer or device
- The Raspberry Pi will act as a remote device: you can connect to it using a client on another machine.

1. Set up your local network and wireless connectivity

- Make sure your Raspberry Pi is properly set up and connected.
- If you are using wireless networking, this can be enabled via the desktop's user interface, or using the command line
- You will need to note down the IP address of your Pi in order to connect to it later.
- Using the ifconfig command will display information about the current network status, including the IP address, or you can use hostname -I to display the IP addresses associated with the device

2. Enable SSH

- Enter sudo raspi-config in a terminal window
- Select Interfacing Options
- Navigate to and select SSH
- Choose Yes
- Select Ok
- Choose Finish

Alternatively, use systemctl to start the service

- sudo systemctl enable ssh
- sudo systemctl start ssh

3. Enable SSH on a headless Raspberry Pi (add file to SD card on another machine)

- For headless setup, SSH can be enabled by placing a file named ssh, without any extension, onto the boot partition of the SD card from another computer.
- When the Pi boots, it looks for the ssh file. If it is found, SSH is enabled and the file is deleted

4. Set up your client

- SSH is built into Linux distributions and Mac OS. For Windows and mobile devices, third-party SSH clients are available.

5.15) Smart and Connected Cities

An IoT Strategy for Smarter Cities

- Managing a city bears some resemblance to managing a corporate enterprise.
- As the need for efficiency increases, new tools help increase operational efficiency.
- For cities, just as for businesses, digitization transforms the perspective on operations.
- New ideas emerge, bringing different approaches to solving management issues.

Vertical IoT Needs for Smarter Cities

- There are many differing approaches and solutions for city management.
- Allthese solutions typically start at the street level, with sensors that capture data on everything from parking space availability to water purity.
- Data analytics is also used extensively—for example, to reduce crime or improve traffic flows.
- Citizens can use tools to leverage their smart mobile devices, such as to report problems and make recommendations for improving urban life or locate available parking spaces.
- When enabled through connectivity, these smart solutions can have a transformative impact on quality of life.
- A recent Cisco study, as illustrated in Figure 5.11, expects IoT to have the following economic impact over a 10-year period:

Figure 5.11 Key Use Cases for Smart Cities

1) Smart buildings: Smart buildings have the potential to save $100 billion by lowering operating costs by reducing energy consumption through the efficient integration of heating, ventilation, and air-conditioning (HVAC) and other building infrastructure systems.

- Note that the financial gain applies to city budgets only when a building is city owned.
- However, the reduced emissions benefit the city regardless of who owns the buildings.

2) Gas monitoring: Monitoring gas could save $69 billion by reducing meter-reading costs and increasing the accuracy of readings for citizens and municipal utility agencies.

- The financial benefit is obvious for users and utility companies when the utility is managed by the city.
- There are also very important advantages in terms of safety, regardless of who operates the utility.
- In cases of sudden consumption increase, a timely alert could lead to emergency response teams being dispatched sooner, thus increasing the safety of the urban environment.

3) Smart parking: Smart parking could create $41 billion by providing realtime visibility into parking space availability across a city.

- Residents can identify and reserve the closest available space, traffic wardens can identify noncompliant usage, and municipalities can introduce demand based pricing.

4) Water management: Smart water management could save $39 billion connecting household water meters over an IP network to provide remote usage and status information.

- The benefit is obvious, with features such as real-time consumption visibility and leak detection.
- A gate or a pump can be opened and closed remotely and automatically in real time, based on a variety of flow input and output analytics data.

- Vibrations can be measured to detect and predict potential equipment failures.
- Repair teams can be dispatched proactively before equipment failure occurs.

Road pricing: Cities could create $18 billion in new revenues by implementing automatic payments as vehicles enter busy city zones while improving overall traffic conditions.

- Real-time traffic condition data is very valuable and actionable information that can also be used to proactively reroute public transportation services or private users.

Global vs. Siloed Strategies

- The main obstacle in implementing smart solutions in today's traditional infrastructure is the complexity of how cities are operated, financed, regulated and planned.
- Cities attempting to upgrade their infrastructure to match the growing needs of the citizen population often invest in one problem at a time, and they do it independently.
- Even cities using IoT technology break up city assets and service management into silos that are typically unable to communicate or rely on each other.

The independent investment model results in the following problems:

  - Isolation of infrastructure and IT resources
  - No sharing of intelligence and information, such as video feeds and data from sensors.
  - Waste and duplication in investment and effort
  - Difficulty scaling infrastructure management
- All these requirements pose technological challenges, including the following:
  - How do you collect the data? What are the various sources of data, including hardware endpoints and software?
  - How do you make sure that any data collection devices, such as sensors, can be maintained without high costs?
  - Where do you analyze the data? What data do you carry back to the cloud, and what data do you analyze locally?
  - What kind of network connectivity is best suited for each type of data to collect?
  - What kind of power availability and other infrastructure, such as storage, is required?
  - How do you aggregate data from different sources to create a unified view?
  - How do you publish the data and make it available for applications to consume?
  - How do you make the end analysis available to specialized smart city personnel, such as traffic operators, parking enforcement officers, street lighting operators, and so on at their logical decision points?
  - How do you present the long-term analysis to city planners?

5.16) Smart City IoT Architecture

- A smart city IoT infrastructure is a four-layered architecture, as shown in Figure 5-12.
- Data flows from devices at the street layer to the city network layer and connect to the data center layer, where the data is aggregated, normalized, and virtualized.
- The data center layer provides information to the services layer, which consists of the applications that provide services to the city.

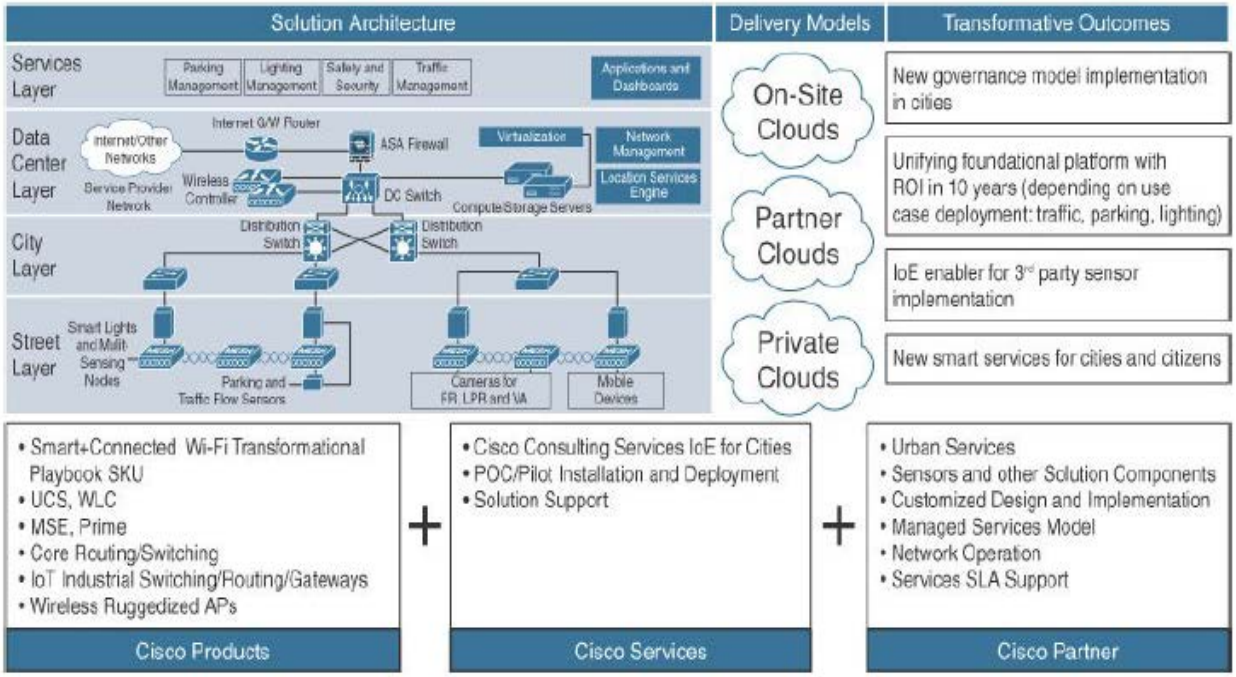


Figure 5.12 Smart Cities Layered Architecture

1) Street Layer

- The street layer is composed of devices and sensors that collect data and take action based on instructions from the overall solution, as well as the networking components needed to aggregate and collect data.
- A sensor is a data source that generates data required to understand the physical world. Sensor devices are able to detect and measure events in the physical world.
- ICT (information and communication technology) connectivity solutions rely on sensors to collect the data from the world around them so that it can be analyzed and used to operationalize use cases for cities.
- A variety of sensors are used at the street layer for a variety of smart city use cases
    - A magnetic sensor can detect a parking event by analyzing changes in the surrounding magnetic field when a heavy metal object, such as a car or a truck, comes close to it (or on top of it).
    - A lighting controller can dim and brighten a light based on a combination of time-based and ambient conditions.

- Video cameras combined with video analytics can detect vehicles, faces, and traffic conditions for various traffic and security use cases.
- An air quality sensor can detect and measure gas and particulate matter concentrations to give a hyper-localized perspective on pollution in a given area.
- Device counters give an estimate of the number of devices in the area, which provides a rough idea of the number of vehicles moving or parked in a street or a public parking area, of pedestrians on a sidewalk, or even of birds in public parks or on public monuments—for cities where bird control has become an issue.

2) City Layer

- At the city layer, which is above the street layer, network routers and switches must be deployed to match the size of city data that needs to be transported.
- This layer aggregates all data collected by sensors and the end-node network into a single transport network.
- The city layer may appear to be a simple transport layer between the edge devices and the data center or the Internet.
- However, one key consideration of the city layer is that it needs to transport multiple types of protocols, for multiple types of IoT applications.
- Some applications are delay- and jitter-sensitive, and some other applications require a deterministic approach to frame delivery.
- As a result, the city layer must be built around resiliency, to ensure that a packet coming from a sensor or a gateway will always be forwarded successfully to the headend station. Figure 5.13 shows a common way of achieving this goal.

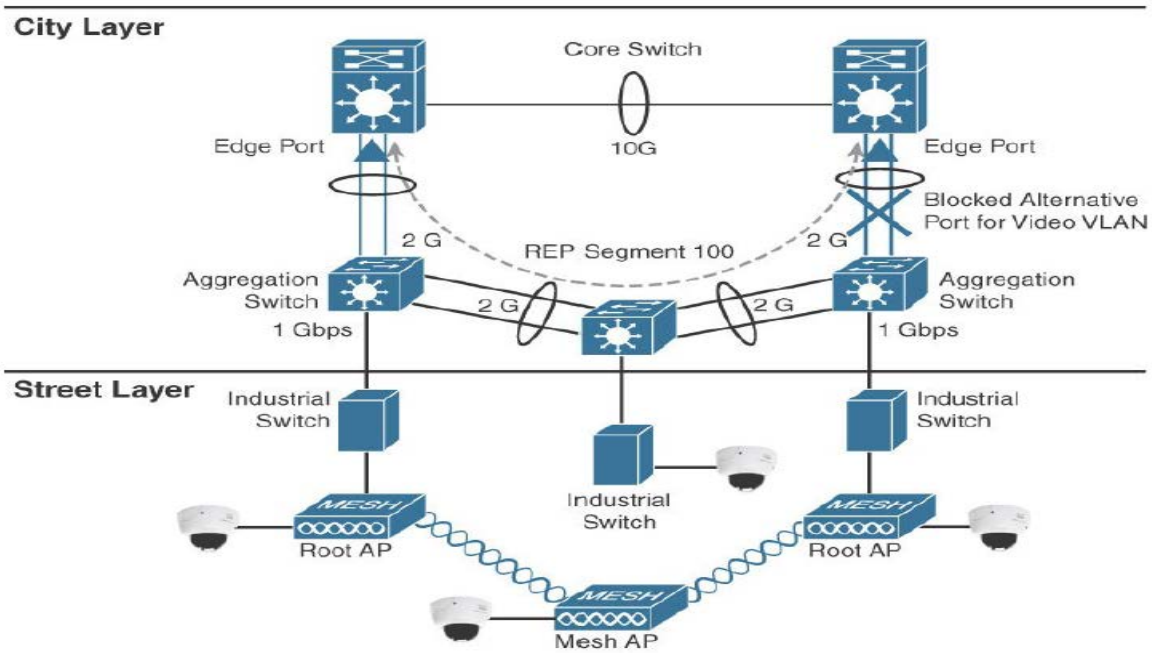


Figure 5.13 Street Layer Resiliency

3) Data Center Layer

- Ultimately, data collected from the sensors is sent to a data center, where it can be processed and correlated.
- Based on this processing of data, meaningful information and trends can be derived, and information can be provided back.
- For example, an application in a data center can provide a global view of the city traffic and help authorities decide on the need for more or less common transport vehicles.
- At the same time, an automated response can be generated.
- For example, the same traffic information can be processed to automatically regulate and coordinate the street light durations at the scale of the entire city to limit traffic congestion.
- The key technology in creating any comprehensive smart solution with services is the cloud.
- With a cloud infrastructure, data is not stored in a data center owned directly or indirectly by city authorities.
- Instead, data is stored in rented logical containers accessed through the Internet.
- Because the containers can be extended or reduced based on needs, the storage size and computing power are flexible and can adapt to changing requirements or budget conditions.
- In addition, multiple contractors can store and process data at the same time, without the complexity of exclusively owned space.
- This proximity and flexibility also facilitate the exchange of information between smart systems and allow for the deployment of new applications that can leverage information from several IoT systems.
- Figure 5.14 shows the vision of utilizing the cloud in smart solutions for cities.
- The cloud provides a scalable, secure, and reliable data processing engine that can handle the immense amount of data passing through it.
- Smart city issues require not just efficient use of infrastructure, which the cloud helps enable, they also require new data processing and management models.
- For example, cloud services allow for Software as a Service (SaaS) models that create cyclical returns on investment.

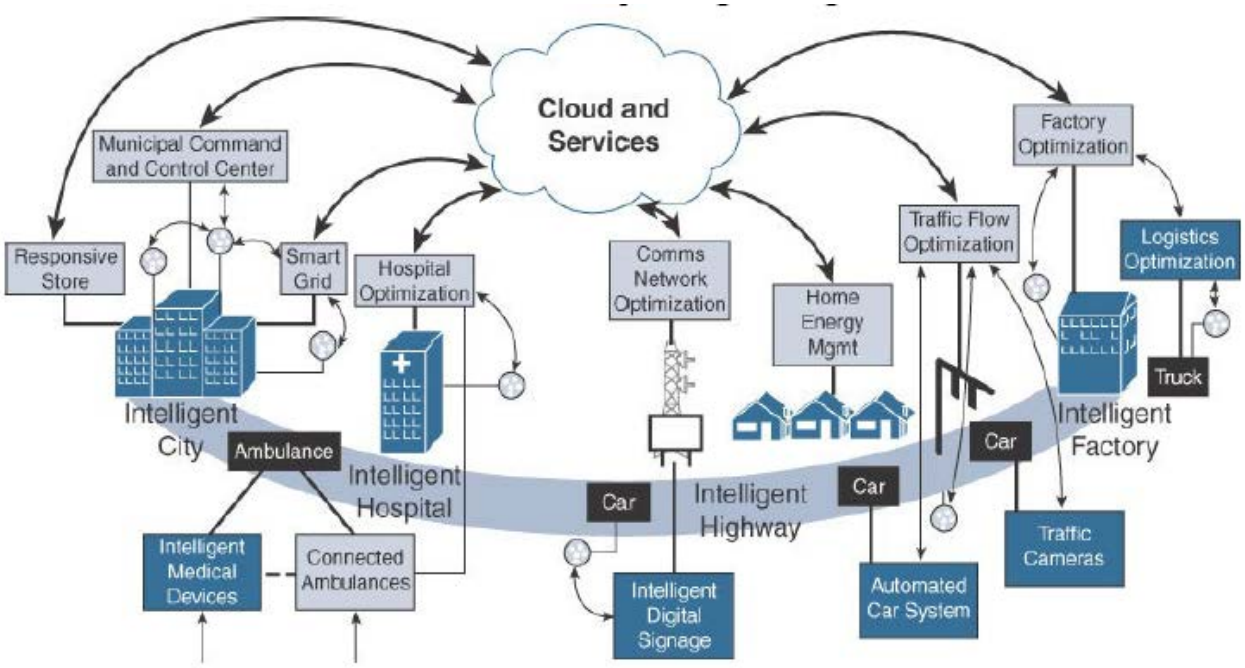


Figure 5.14 The Role of the Cloud for Smart City Applications

- With the cloud approach shown in Figure 5.14 , smart cities can also take advantage of operating expense–based consumption models to overcome any financial hurdles in adopting solutions to their most critical issues.
- Critical data, such as air condition (humidity, temperature, pollution) levels monitoring, can be processed initially.
- Then, as the efficiency of IoT is scaled up, richer data processing can be enabled in the cloud applications.
- For example, the humidity level can be used to regulate the color and luminosity of street lights. In times when city budgets are strained, data processing can be scaled down to essential services.

4) Services Layer

- Ultimately, the true value of ICT connectivity comes from the services that the measured data can provide to different users operating within a city.
- Smart city applications can provide value to and visibility for a variety of user types, including city operators, citizens, and law enforcement.
- The collected data should be visualized according to the specific needs of each consumer of that data and the particular user experience requirements and individual use cases.
- For example, parking data indicating which spots are and aren't currently occupied can drive a citizen parking app with a map of available spots, as well as an enforcement officer's understanding of the state (utilization and payment) of the public parking space,

while at the same time helping the city operator's perspective on parking problem areas in the city at any given time.

- With different levels of granularity and scale, the same data performs three different functions for three different users.
- Along the same lines, traffic information can be used by individual car drivers to find the least congested route.
- A variation of the same information can be made available to public transportation users to estimate travel times.
- Public transportation systems, such as buses, can be rerouted around known congestion points.
- The number of subway trains can be increased dynamically to respond to an increase in traffic congestion, anticipating the decisions of thousands or even millions of commuters to take public transportation instead of cars on days when roads are very congested.
- Here again, the same type of data is utilized by different types of users in different ways based on their specific use cases.

On-Premises vs. Cloud

- Different cities and regions have different data hosting requirements based on security or legal policies. A key consideration in developing ICT connectivity solutions is whether a city has requirements about where data should be hosted.
- Data can be hosted on-premises or in the cloud. Fog architectures provide an intermediate layer.
- The data resulting from fog processing can be sent to the cloud or to a data center operated locally (on-premises).
- On-premises encompasses traditional networks, and all their limitations, whereas cloud hosting encompasses a whole host of security risks if the proper measures are not taken to secure citizen data.
- When data is sent to the cloud, data sovereignty (supremacy) laws may restrict the physical location where this data is actually stored.
- Ideally, a smart city utilizing ICT connectivity would use the cloud in its architecture, but if this is impossible, the city would need to invest far more in the city layer's networking components (for example, switches, routers) and still may not be able to drive the same cross-domain value propositions and scalability in its design.

5.17) Smart City Security Architecture

- A serious concern of most smart cities and their citizens is data security.
- Vast quantities of sensitive information are being shared at all times in a layered, realtime architecture, and cities have a duty to protect their citizens' data from unauthorized access, collection, and tampering.
- In general, citizens feel better about data security when the city itself, and not a private entity, owns public or city-relevant data.
- It is up to the city and the officials who run it to determine how to utilize this data.
- When a private entity owns city-relevant data, the scope of the ownership may initially be very clear.
- However, later considerations or changes in the private entity strategy may shift the way the data is used.
- It may then be more difficult for city authorities or the citizens to oppose this new direction, simply because they do not have any stake in the decision-making process of the private entity.
- For example, suppose that a private contractor is in charge of collecting and managing parking sensor data.
- One possible way to increase the profitability of such data is to sell it to insurance companies looking to charge an additional premium to car owners parking in the street (vs. in a covered and secured garage).
- Such deviations from the original mandate are less likely to happen when cities own the data and when citizens have a way to vote against such usages.
- A security architecture for smart cities must utilize security protocols to fortify each layer of the architecture and protect city data.
- Figure 5.15 shows a reference architecture, with specific security elements highlighted.

Security protocols should authenticate the various components and protect data transport throughout.

- For example, hijacking traffic sensors to send false traffic data to the system regulating the street lights may result in dramatic congestion issues.
- The benefit for the offender may be the ability to get "all greens" while traveling, but the overall result would typically be dangerous and detrimental to the city.
- The security architecture should be able to evolve with the latest technology and incorporate regional guidelines (for example, city by-laws, county or regional security regulations).

- Network partners may also have their own compliance standards, security policies, and governance requirements that need to be added to the local city requirements.

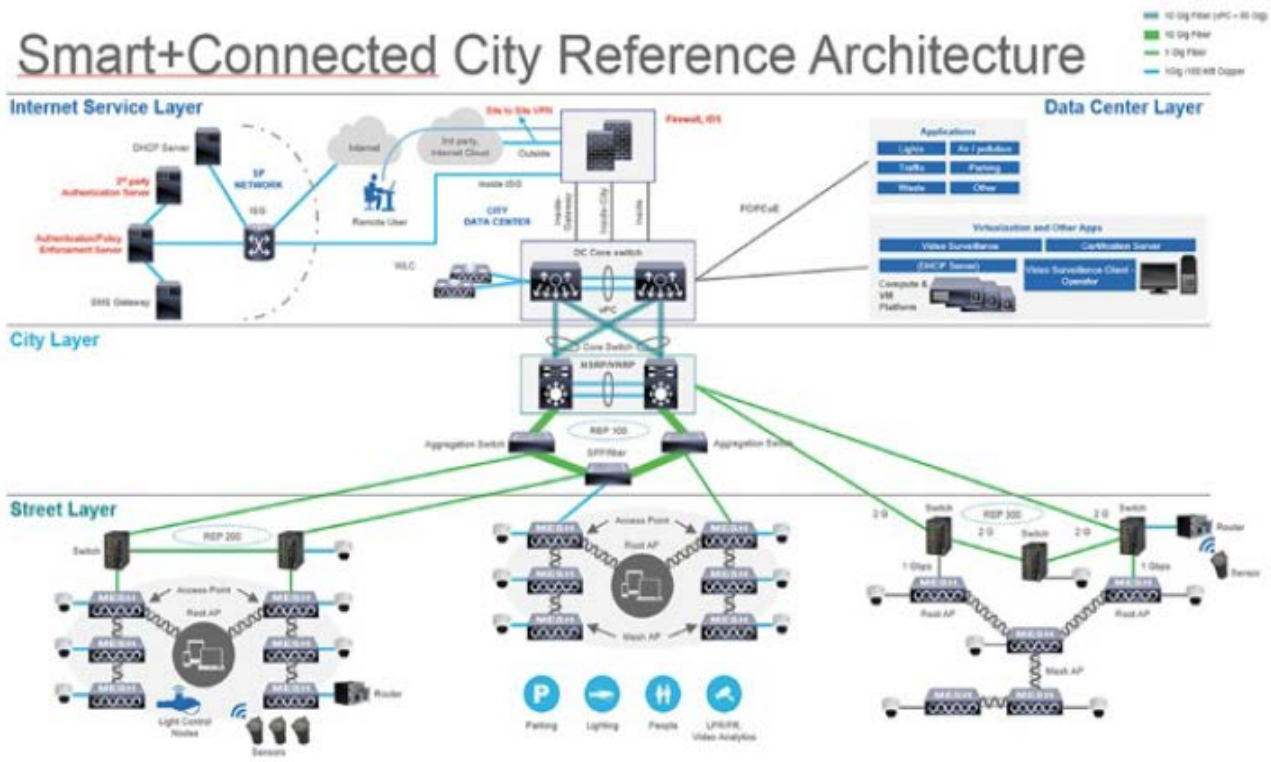


Figure 5.15 Key Smart and Connected Cities Reference Architecture

The following are common industry elements for security on the network layer:

- **Firewall:** A firewall is located at the edge, and it should be IPsec- and VPN-ready, and include user- and role-based access control.
- It should also be integrated with the architecture to give city operators remote access to the city data center.
- **VLAN:** A VLAN provides end-to-end segmentation of data transmission, further protecting data from rogue intervention. Each service/domain has a dedicated VLAN for data transmission.
- **Encryption:** Protecting the traffic from the sensor to the application is a common requirement to avoid data tampering and eavesdropping.
- In most cases, encryption starts at the sensor level. In some cases, the sensor-to gateway link uses one type of encryption, and the gateway-to-application connection uses another encryption (for example, a VPN).

5.17) Smart City Use-Case Examples

- There are multiple ways a smart city can improve its efficiency and the lives of its citizens.
- The following sections examine some of the applications commonly used as starting

points to implement IoT in smart cities: connected street lighting, smart parking, smart traffic control, and connected environment.

Connected Street Lighting

- Of all urban utilities, street lighting comprises one of the largest expenses in a municipality's utility bill, accounting for up to 40% of the total, according to the New York State Department of Environmental Conservation.
- Maintenance of street lights is an operational challenge, given the large number of lights and their vast geographic distribution.

Connected Street Lighting Solution

- Cities commonly look for solutions to help reduce lighting expenses and at the same time improve operating efficiencies while minimizing upfront investment.
- The installation of a smart street lighting solution can provide significant energy savings and can also be leveraged to provide additional services.
- In this regard, light-emitting diode (LED) technology leads the transition from traditional street lighting to smart street lighting:
- LEDs require less energy to produce more light than legacy lights, and they have a much longer life span and a longer maintenance cycle.
- A leading lighting company estimates that a complete switch to LED technology can reduce individual light bills by up to 70%.
- LEDs are well suited to smart solution use cases.
- For example, LED color or light intensity can be adapted to site requirements (for example, warmer color and lower intensity in city centers, sun-like clarity on highways, time- and weather-adaptive intensity and color).
- Figure 5.16 shows how electricity prices rise, while LED prices decrease and their unit sales rise.

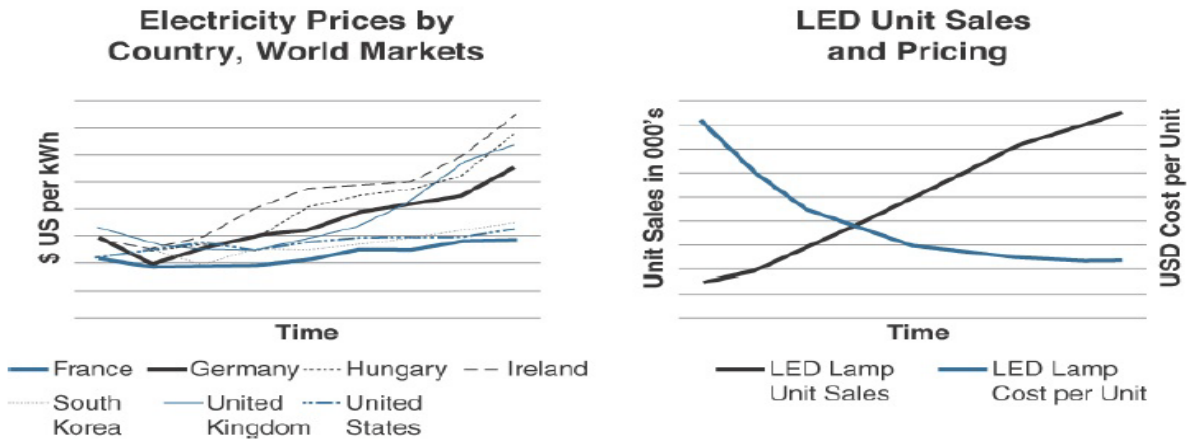


Figure 5.16 Electricity Cost vs. LED Cost and Sales

Street Lighting Architecture

- Connected lighting uses a light management application to manage street lights remotely by connecting to the smart city's infrastructure.
- This application attaches to LED lights, monitors their management and maintenance, and allows you to view the operational status of each light.
- In most cases, a sensor gateway acts as an intermediate system between the application and the lights (light control nodes).
- The gateway relays instructions from the application to the lights and stores the local lights' events for the application's consumption.
- The controller and LED lights use the cloud to connect to the smart city's infrastructure, as shown in Figure 5.17.

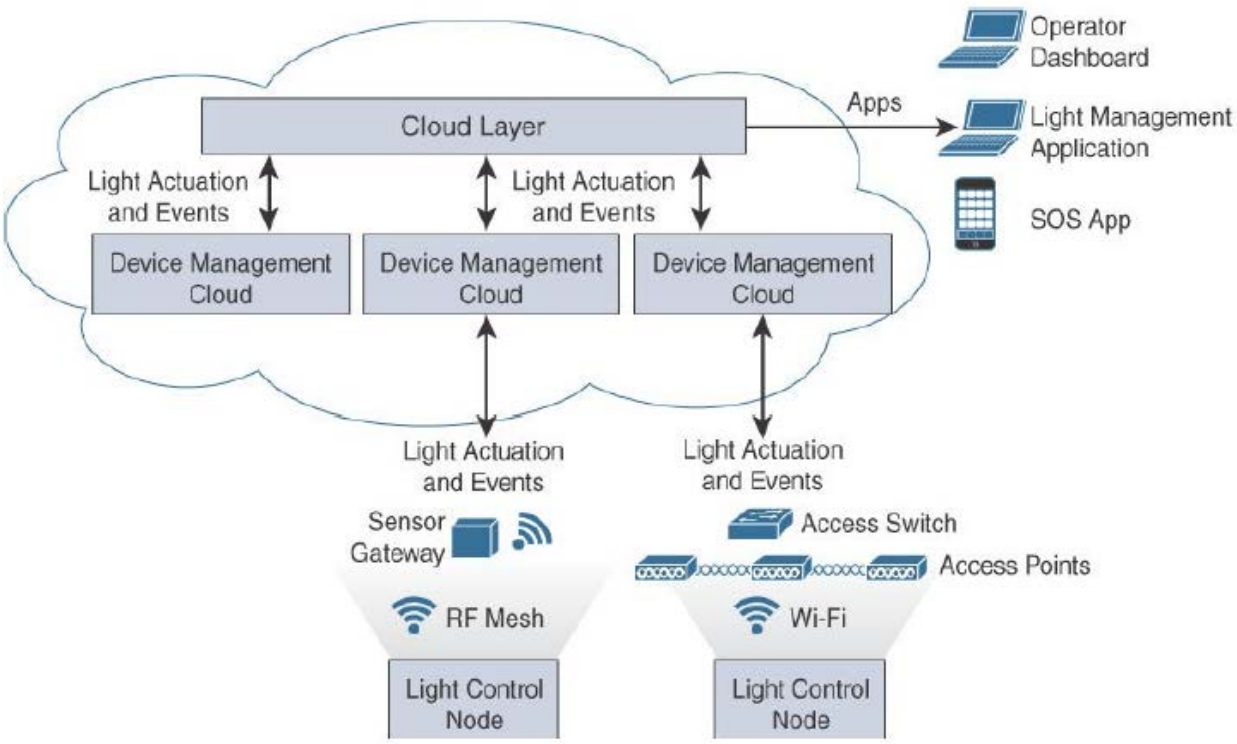


Figure 5.17 Connected Lighting Architecture

- A human or automated operator can use a cloud application to perform automated scheduling for lights and even get light sensors to perform automated dimming or brightening, as needed.
- The schedule can also impact the light intensity level and possibly the color, depending on environmental conditions, weather, time of year, time of day, location within the city, and so on.

- Lighting nodes vary widely in the industry, especially with respect to elements such as what communication protocol they use (for example, Wi-Fi, cellular, ZigBee, 802.15.4g [Wi-SUN], LoRaWAN).
- These features are optimized for different circumstances and conditions; no single lighting node can support all environments ideally.
- Many solutions leverage wired connectivity, either by using the existing city cable infrastructure or by adding a cable adjacent to the power cable.
- In cases where cabling is not practical, wireless technologies may bring interesting capabilities.
- For example, 802.15.4g controllers can be used to form a mesh and extend the network. This extension is used not only to connect other light poles but also to connect smart meters from neighboring houses.
- In all cases, the built-in versatility offered by the four-layer architecture shown in Figure 5.17 ensures that all the different types of technologies optimized to fit any city topology can be flexibly incorporated into the solution.

Smart Parking Use Cases

- Added traffic congestion is one consequence of drivers looking for parking space, and it has several consequences:
- **Contributes to pollution:** Tons of extra carbon emissions are released into the city's environment due to cars driving around searching for parking spots when they could be parked.
- **Causes motorist frustration:** In most cities, parking spot scarcity causes drivers to lose patience and waste time, leading to road rage, inattention, and other stress factors.
- **Increases traffic incidents:** Drivers searching for parking spots cause increased congestion in the streets and that, in turn, causes increased accidents and other traffic incidents.
- Revenue loss is another consequence of drivers looking unsuccessfully for parking space, and it also has various negative side effects:
- **Cities often lose revenue:** As a result of inadequate parking meter enforcement and no-parking, no-standing, and loading-zone violations, cities lose revenue.
- **Parking administration employee productivity suffers:** Employees waste time roaming the streets, attempting to detect parking rules offenders.
- **Parking availability affects income:** Local shops and businesses lose customers because of the decreased accessibility caused by parking space shortages.

Smart Parking Architecture

- A variety of parking sensors are available on the market, and they take different approaches to sensing occupancy for parking spots.
- Examples include in-ground magnetic sensors, which use embedded sensors to create a magnetic detection field in a parking spot; video-based sensors, which detect events based on video computing (vehicle movements or presence); and radar sensors that sense the presence of vehicles (volumetric detection).
- Most sensors installed in the ground must rely on battery power, since running a power line is typically too expensive.
- In larger (for example, outdoor) environments, a longer-range Low Power Wide Area (LPWA) protocol is common, as shown in Figure 5.18.

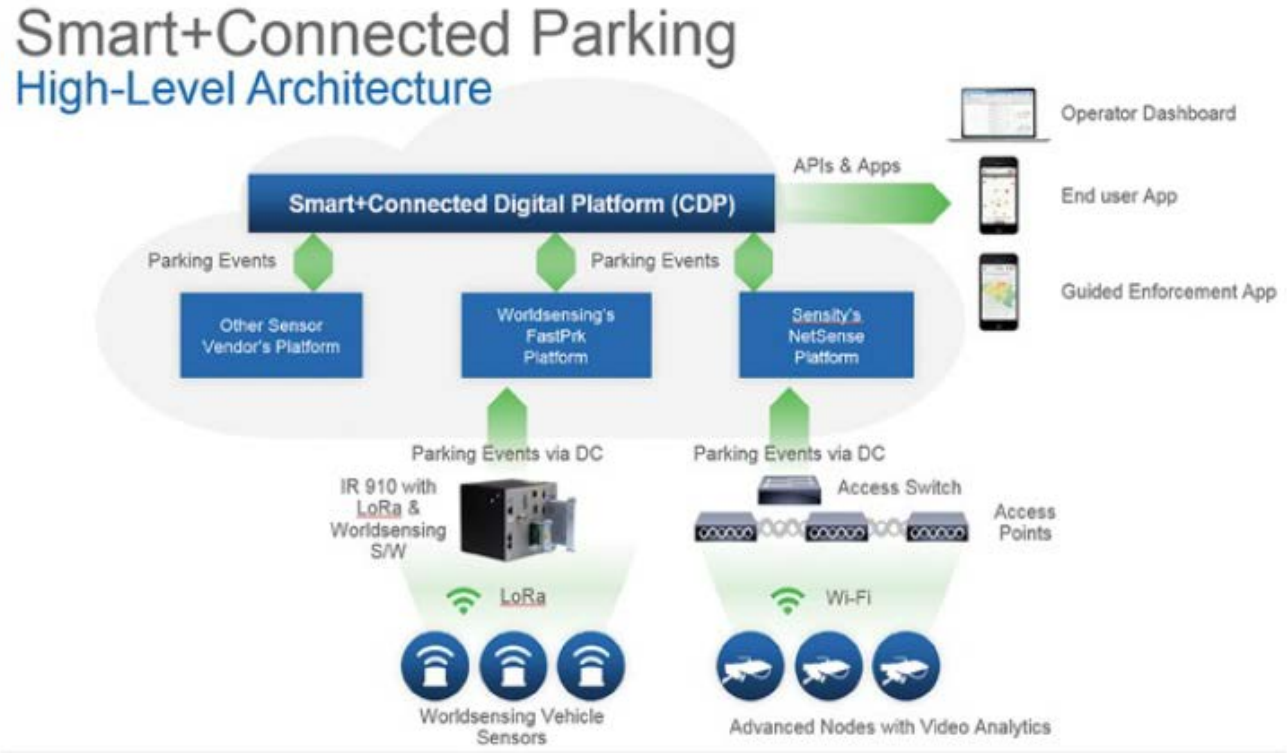


Figure 5.18 Connected Parking Architecture

- Parking sensors are typically event-driven objects.
- A sensor detects an event and identifies it based on time or analysis.
- The event is transmitted through the device's communication protocol to an access point or gateway, which forwards the event data through the city layer.
- The gateway sends it to the cloud or a fog application, where it is normalized.
- An application shows the parking event on operator dashboards, or personal smart phones, where an action can be taken.

- For example, a driver can book a nearby parking spot, or a parking operator can remove it from the list of available parking spaces in target locations.
- This action triggers data to be sent back to the parking sensor to modify its availability status based on the received instructions.
- smart parking has three users that applications must support through aggregated data: city operators, parking enforcement personnel, and citizens.
- The true value of data normalization is that all parking data, regardless of technology or vendor, would be visible in these applications for the different users to support their particular experiences.
- The following are some potential user experiences for these three user types.

1) City operators: These users might want a high-level map of parking in the city to maintain perspective on the city's ongoing parking situation.

- They would also need information on historical parking data patterns to understand congestion and pain points in order to be able to effectively influence urban planning.

2) Parking enforcement officers: These users might require real-time updates on parking changes in a certain area to be able to take immediate action on enforcement activities, such as issuing tickets or sending warnings to citizens whose time is nearing expiration.

- Their focus is driving revenue creation for the city and minimizing wasted time by performing parking monitoring and enforcement at scale

3) Citizens: These users might want an application with a map (such as a built-in parking app in their car) showing available parking spots, reservation capabilities, and online payment.

- Their focus would be on minimizing the time to get a parking spot and avoiding parking tickets.
- The application could warn when parking duration limits approach, allowing the driver to move the vehicle before the timer expires or pay a parking timer extension fee without having to go back to the vehicle.

Smart Traffic Control

- Traffic is one the most well-understood pain points for any city.
- It is the leading cause of accidental death globally, causes immense frustration, and heavily contributes to pollution around the globe.
- A smart city traffic solution would combine crowd counts, transit information, vehicle counts, and so on and send events regarding incidents on the road so that other controllers on the street could take action.

Smart Traffic Control Architecture

- In the architecture shown in Figure 5.19, a video analytics sensor computes traffic events based on a video feed and only pushes events (the car count, or metadata, not the individual images) through the network.
- These events go through the architectural layers and reach the applications that can drive traffic services.
- These services include traffic light coordination and also license plate identification for toll roads.
- Some sensors can also recognize abnormal patterns, such as vehicles moving in the wrong direction or a reserved lane. In that case, the video feed itself may be uploaded to traffic enforcement agencies.

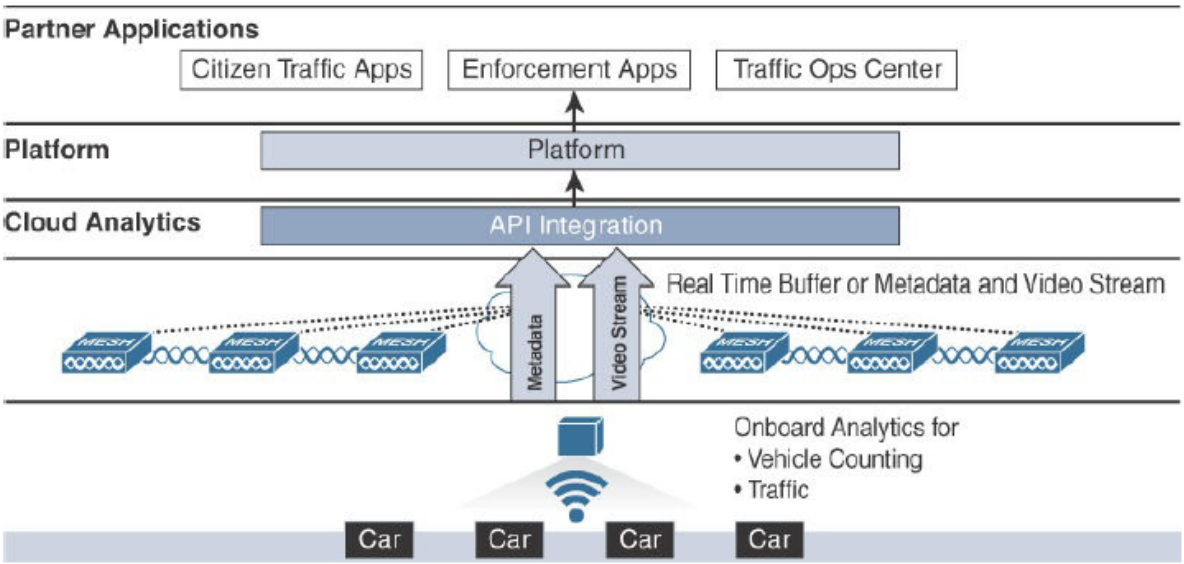


Figure 5.19 Smart City Traffic Architecture

- Other types of sensors that are part of traffic control solutions include Bluetooth vehicle counters, real-time speed and vehicle counters, and lighting control systems.
- These sensors provide a real-time perspective while also offering data collection services for historical data trending and correlation purposes.

Smart Traffic Applications

- Traffic applications can be enabled to take immediate action with other sensors to manage traffic and to reduce pain points. Historical data can be used to develop more efficient urban planning to reduce the amount of traffic a city experiences.
- A common traffic pain point is stop-and-go, where traffic flow suddenly comes to a halt and then flows again.
- This wavelike traffic pattern is a natural result of the unpredictability of the traffic speed ahead and has long been studied by public and private organizations.

- A well-known remedy for stop-and-go traffic is to regulate the standard flow speed based on car density.
- As density increases, car speed is forced down to avoid the wave effect.
- An application that measures traffic density in real time can take action by regulating the street light cycle duration to control the number of cars added to the flow of the main routes, thus limiting or suppressing the wave effect.
- From the driver's standpoint, there is a wait time before being able to get on the highway or main street, and traffic on the main route is slow but steady.
- The impression is that traffic is slow but moving, and the overall result is a better commute experience, with lowered and less stressful commute time, as well as a reduced number of accidents.

Connected Environment

- As of 2017, 50% of the world's population has settled on less than 2% of the earth's surface area.
- Such densely populated closed spaces can see spikes in dangerous gas molecules at any given moment.
- More than 90% of the world's urban population breathes in air with pollutant levels that are much higher than the recommended thresholds, and one out of every eight deaths worldwide is a result of polluted air.

The Need for a Connected Environment

- Most large cities monitor their air quality.
- Data is often derived from enormous air quality monitoring stations that are expensive and have been around for decades.
- These stations are highly accurate in their measurements but also highly limited in their range, and a city is likely to have many blind spots in coverage.
- Given the price and size of air quality monitoring stations, cities cannot afford to purchase the number of stations required to give accurate reports on a localized level and follow the pollution flows as they move through the city over time.
- To fully address the air quality issues in the short term and the long term, a smart city would need to understand air quality on a hyper-localized, real-time, distributed basis at any given moment. To get those measurements, smart cities need to invest in the following:
    - Open-data platforms that provide current air quality measurements from existing air quality monitoring stations

- Sensors that provide similar accuracy to the air quality stations but are available at much lower prices
- Actionable insights and triggers to improve air quality through cross domain actions
- Visualization of environmental data for consumers and maintenance of historical air quality data records to track emissions over time.

Connected Environment Architecture

- Figure 5.20 shows an architecture in which all connected environment elements overlay on the generalized four-layer smart city IoT architecture.

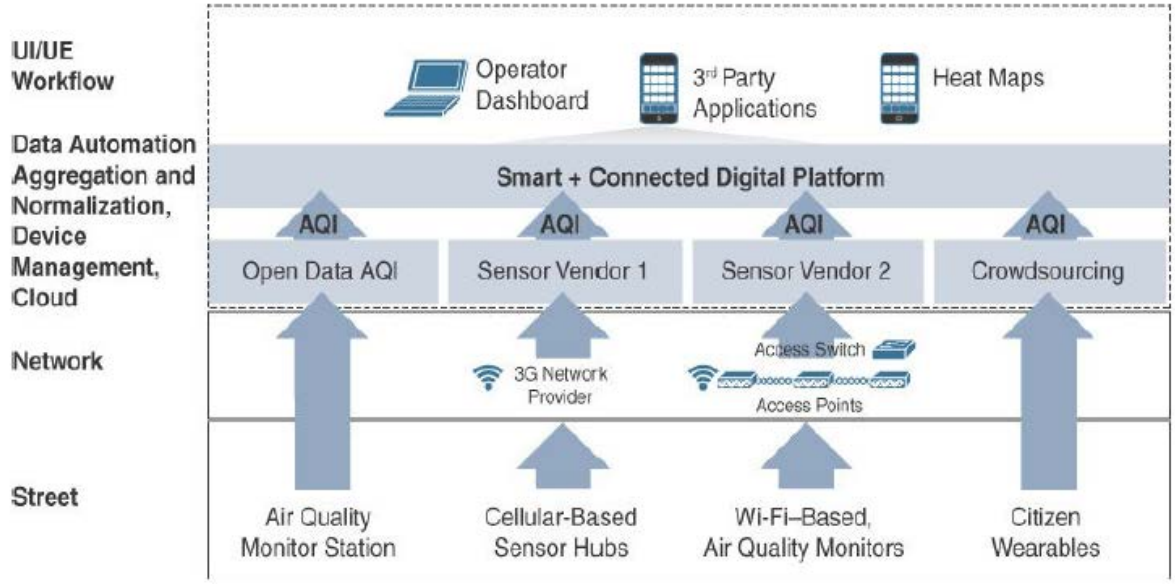


Figure 5.20 Connected Environment Architecture

- As shown in Figure 5.20 , at the street layer there are a variety of multivendor sensor offerings, using a variety of communication protocols.
- Connected environment sensors might measure different gases, depending on a city's particular air quality issues, and may include weather and noise sensors.
- These sensors may be located in a variety of urban fixtures, such as in street lights, as explained earlier.
- They may also be embedded in the ground or in other structures or smart city infrastructure.
- Even mobile sources of information can be included through connected wearables that citizens might choose to purchase and carry with them to understand the air quality around them at any given moment.
- Crowd sourcing may make this information available to the global system.
- Communication technologies depend on the location of the sensors.

- Wearables typically communicate via a short-range technology (such as Bluetooth) with a nearby collecting device (such as a phone).
- That device, in turn, forwards the collected data to the infrastructure (for example, through cellular data).
- Sensors that are installed in urban fixtures also use a variety of communication technologies.
- Sensors included in street lighting systems may utilize the same communication infrastructure as the street light control application.
- In addition to all the air quality sensor and wearable data, the data center layer or application layer represented on the left side of Figure 5.20 also receives the open data from existing weather stations as an additional data input.
- All these data inputs come together to provide a highly accurate sense of the air quality in the city at any given moment.

Model Questions

QE1) Compare digital and analog temperature sensor

QE2) What is thermistor? What is its role in temperature sensor?

1) What are the things Arduino can do?

2) Explain Arduino UNO Board.

or

Explain Arduino UNO microcontroller with neat sketch

3) Explain how arduino software package is downloading and installation (write steps)

4) List Technical Specification of Arduino UNO

5) Explain the Data types , operators and constants in Arduino programming.

6) Explain Flow control statements in Arduino programming.

7) Explain Digital and Analog input output pins and their usage with respect to Arduino

8) Compare Raspberry Pi model A+, Model B, Model B+ and Model 3.

9) Draw a neat diagram for Raspberry Pi GPIO pins .

10) Explain Raspberry Pi learning Board.

11) Explain configuring RaspberryPi microcontroller.

12) Explain different categories of Raspberry operating system.

13) Write a python program to check for Armstrong number

14) Describe DS18B20 Temperature Sensor.

15) Give technical specifications of DS18B20 Temperature Sensor along with PIN Diagram

16) Explain Pi via SSH.

17) Write a program to access temperature from DS18B20 sensor.

or

Write a python code to read temperature value in centigrade (Celsius) and Fahrenheit form from DS18B20 sensor connected to Raspberry pi microcontroller.

18) Explain Smart City IoT Architecture with a neat diagram.

19) Explain Smart City Security Architecture with a neat diagram.