

# EMBEDDED GPU LAB MANUAL

S. No	Experiment Name	Pg. No
1	GPIO programming (LED Blinking) on Jetson Nano	2
2	Interfacing sensors and actuators to Jetson Nano.	3
3	Interfacing camera and modules with Jetson Nano.	6
4	To perform data classification using Jetson Nano.	7
5	Write a CUDA program to demonstrate squaring an array using CUDA kernel.	8
6	Write a CUDA C program to add two large vectors.	-
7	Design parallel algorithm for matrix multiplication using CUDA.	-
8	Write a CUDA program to find out minimum among 100 values using a CUDA kernel.	-
9	Write an OpenCL program for matrix multiplication.	-
10	Write an OpenCL program for calculating value of $\pi$	-

**NOTE: Common Requirements for all the Experiments include Jetson Nano CPU(GPU), Display Monitor, Mouse, Keyboard and Power Adapter of Jetson Nano.**

## Pre-Checks:

1. The Nvidia Jetson Nano GPU is to be pre-loaded with Nvidia Ubuntu 18.04 LTS Operating System.
2. Hereby, All the Experiments are performed on real time Nvidia Ubuntu 18.04 LTS Operating System.
3. Make sure the short pins on Jetson Nano GPU are shunt using Standard Header Pin Jumper Cap. Without this, the Jetson Nano will not be able to work.
4. For further information, please refer to <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit#prepare>

## EXPERIMENT 1: GPIO programming (LED Blinking) on Jetson Nano.

**AIM:** To Interface LED with Jetson Nano and Control it using GPIO Programming.

**Requirements:** Common Requirements, LEDs, *Jetson.GPIO* (Library from pip3)

CODE:

```
import Jetson.GPIO as GPIO
import time
led_pin = 7
GPIO.setmode(GPIO.BOARD)
GPIO.setup(led_pin, GPIO.OUT, initial=GPIO.HIGH)

while True:
    time.sleep(2)
    GPIO.output(led_pin, GPIO.HIGH)
    print("LED is ON")
    time.sleep(2)
    GPIO.output(led_pin, GPIO.LOW)
    print("LED is OFF")
```

**NOTE:** Connect LED to Pin 7 on Jetson Nano Board.

**Result:** You can See LED Blinking.

**Terminal OUTPUT:**

A screenshot of a terminal window with a dark background. It displays a series of 16 lines of text, alternating between "LED is ON" and "LED is OFF". The text is in a light-colored, monospaced font. The sequence starts with "LED is ON" and ends with "LED is OFF".

```
LED is ON
LED is OFF
LED is ON
LED is OFF
LED is ON
LED is OFF
LED is ON
LED is OFF
LED is ON
LED is OFF
LED is ON
LED is OFF
LED is ON
LED is OFF
LED is ON
LED is OFF
```

**EXPERIMENT 2:** Interfacing sensors and actuators to Jetson Nano.

**AIM:** To Interfacing MPU6050 to Jetson Nano using *SMBUS* library.

**Requirements:** Common Requirements, MPU6050, *SMBUS* (Library from python3)

**CODE:**

```
import smbus
from time import sleep

#some MPU6050 Registers and their Address
PWR_MGMT_1    = 0x6B
SMPLRT_DIV    = 0x19
CONFIG        = 0x1A
GYRO_CONFIG   = 0x1B
INT_ENABLE    = 0x38
ACCEL_XOUT_H  = 0x3B
ACCEL_YOUT_H  = 0x3D
ACCEL_ZOUT_H  = 0x3F
GYRO_XOUT_H   = 0x43
GYRO_YOUT_H   = 0x45
GYRO_ZOUT_H   = 0x47

def MPU_Init():
    #write to sample rate register
    bus.write_byte_data(Device_Address, SMPLRT_DIV, 7)

    #Write to power management register
    bus.write_byte_data(Device_Address, PWR_MGMT_1, 1)

    #Write to Configuration register
    bus.write_byte_data(Device_Address, CONFIG, 0)

    #Write to Gyro configuration register
    bus.write_byte_data(Device_Address, GYRO_CONFIG, 24)

    #Write to interrupt enable register
    bus.write_byte_data(Device_Address, INT_ENABLE, 1)

def read_raw_data(addr):
    #Accelero and Gyro value are 16-bit
    high = bus.read_byte_data(Device_Address, addr)
    low = bus.read_byte_data(Device_Address, addr+1)

    #concatenate higher and lower value
    value = ((high << 8) | low)

    #to get signed value from mpu6050
    if(value > 32768):
```

```

        value = value - 65536
    return value

bus = smbus.SMBus(1)    # or bus = smbus.SMBus(0) for older version boards
Device_Address = 0x68    # MPU6050 device address

MPU_Init()

print (" Reading Data of Gyroscope and Accelerometer")

while True:

    #Read Accelerometer raw value
    acc_x = read_raw_data(ACCEL_XOUT_H)
    acc_y = read_raw_data(ACCEL_YOUT_H)
    acc_z = read_raw_data(ACCEL_ZOUT_H)

    #Read Gyroscope raw value
    gyro_x = read_raw_data(GYRO_XOUT_H)
    gyro_y = read_raw_data(GYRO_YOUT_H)
    gyro_z = read_raw_data(GYRO_ZOUT_H)

    #Full scale range +/- 250 degree/C as per sensitivity scale factor
    Ax = acc_x/16384.0
    Ay = acc_y/16384.0
    Az = acc_z/16384.0

    Gx = gyro_x/131.0
    Gy = gyro_y/131.0
    Gz = gyro_z/131.0

    print ("Gx=%.2f" %Gx, u'\u00b0'+ "/s", "\tGy=%.2f" %Gy, u'\u00b0'+
"/s", "\tGz=%.2f" %Gz, u'\u00b0'+ "/s", "\tAx=%.2f g" %Ax, "\tAy=%.2f g"
%Ay, "\tAz=%.2f g" %Az)
    sleep(1)

```

**NOTE:** Connect VCC to pin 17, GND to pin GND, SCL to pin 5, SDA to pin 3 on Jetson Nano Board.

**Terminal OUTPUT:**

```
Reading Data of Gyroscope and Accelerometer
Gx=0.27 °/s   Gy=-0.60 °/s   Gz=-0.18 °/s   Ax=0.17 g   Ay=0.58 g   Az=-0.87 g
Gx=-0.07 °/s  Gy=-0.47 °/s   Gz=-0.64 °/s   Ax=0.15 g   Ay=0.58 g   Az=-0.91 g
Gx=0.06 °/s   Gy=-0.66 °/s   Gz=-0.47 °/s   Ax=0.18 g   Ay=0.57 g   Az=-0.86 g
Gx=-0.12 °/s  Gy=-0.56 °/s   Gz=-0.55 °/s   Ax=0.15 g   Ay=0.58 g   Az=-0.88 g
Gx=0.08 °/s   Gy=-0.53 °/s   Gz=-0.44 °/s   Ax=0.19 g   Ay=0.56 g   Az=-0.86 g
Gx=-0.21 °/s  Gy=-0.50 °/s   Gz=-0.62 °/s   Ax=0.11 g   Ay=0.59 g   Az=-0.88 g
Gx=0.15 °/s   Gy=-0.76 °/s   Gz=-0.59 °/s   Ax=0.17 g   Ay=0.59 g   Az=-0.88 g
Gx=-0.31 °/s  Gy=-0.18 °/s   Gz=-0.43 °/s   Ax=0.13 g   Ay=0.59 g   Az=-0.91 g
Gx=-0.01 °/s  Gy=-0.89 °/s   Gz=-0.83 °/s   Ax=0.17 g   Ay=0.55 g   Az=-0.86 g
Gx=-0.11 °/s  Gy=-0.45 °/s   Gz=0.06 °/s    Ax=0.18 g   Ay=0.62 g   Az=-0.85 g
Gx=-0.05 °/s  Gy=-0.47 °/s   Gz=-0.37 °/s   Ax=0.21 g   Ay=0.54 g   Az=-0.89 g
Gx=-0.13 °/s  Gy=-0.57 °/s   Gz=-0.17 °/s   Ax=0.15 g   Ay=0.60 g   Az=-0.91 g
Gx=0.00 °/s   Gy=-0.36 °/s   Gz=-0.44 °/s   Ax=0.17 g   Ay=0.55 g   Az=-0.90 g
Gx=-0.15 °/s  Gy=-0.56 °/s   Gz=-0.45 °/s   Ax=0.16 g   Ay=0.62 g   Az=-0.88 g
Gx=-0.19 °/s  Gy=-0.25 °/s   Gz=0.21 °/s    Ax=0.16 g   Ay=0.62 g   Az=-0.91 g
Gx=0.02 °/s   Gy=-0.67 °/s   Gz=-0.84 °/s   Ax=0.16 g   Ay=0.55 g   Az=-0.86 g
Gx=-0.16 °/s  Gy=-0.58 °/s   Gz=-0.33 °/s   Ax=0.17 g   Ay=0.57 g   Az=-0.96 g
Gx=0.11 °/s   Gy=-0.63 °/s   Gz=-0.31 °/s   Ax=0.16 g   Ay=0.58 g   Az=-0.88 g
Gx=-0.10 °/s  Gy=-0.71 °/s   Gz=-0.42 °/s   Ax=0.16 g   Ay=0.58 g   Az=-0.86 g
Gx=0.02 °/s   Gy=-0.56 °/s   Gz=-0.34 °/s   Ax=0.15 g   Ay=0.59 g   Az=-0.90 g
Gx=-0.26 °/s  Gy=-0.47 °/s   Gz=-0.38 °/s   Ax=0.17 g   Ay=0.57 g   Az=-0.88 g
Gx=0.10 °/s   Gy=-0.57 °/s   Gz=-0.31 °/s   Ax=0.16 g   Ay=0.58 g   Az=-0.89 g
Gx=-0.08 °/s  Gy=-0.56 °/s   Gz=-0.37 °/s   Ax=0.15 g   Ay=0.56 g   Az=-0.90 g
Gx=0.12 °/s   Gy=-0.55 °/s   Gz=-0.35 °/s   Ax=0.16 g   Ay=0.56 g   Az=-0.89 g
Gx=-0.10 °/s  Gy=-0.44 °/s   Gz=-0.37 °/s   Ax=0.17 g   Ay=0.56 g   Az=-0.89 g
Gx=-0.53 °/s  Gy=-0.53 °/s   Gz=-0.27 °/s   Ax=0.14 g   Ay=0.57 g   Az=-0.90 g
Gx=-0.53 °/s  Gy=-0.53 °/s   Gz=-0.27 °/s   Ax=0.16 g   Ay=0.57 g   Az=-0.89 g
```

**EXPERIMENT 3:** Interfacing camera and modules with Jetson Nano.

**AIM:** To Interfacing camera and modules with Jetson Nano.

**Requirements:** Common Requirements, Camera, *Opencv2* (Python3 Library)

**CODE:**

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)
while(cap.isOpened()):

    while True:

        ret, img = cap.read()
        cv2.imshow('img', img)
        if cv2.waitKey(30) & 0xff == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
else:
    print("Alert ! Camera disconnected")
```

**NOTE:** Connect the USB Camera to Jetson Nano Board.

**Terminal Output:** One Can observe Live Video Steam.

**EXPERIMENT 4:** To perform data classification using Jetson Nano.

**AIM:** To perform data classification using Jetson Nano.

**TO Be Updated!!!**

**EXPERIMENT 5:** Write a CUDA program to demonstrate squaring an array using CUDA kernel.

**AIM:** To Write a CUDA program to demonstrate squaring an array using CUDA kernel.

**Requirements:** Common Requirements.

**CODE:**

```
#include <iostream>
```

```
#include <numeric>
```

```
#include <stdlib.h>
```

```
#include <cuda.h>
```

```
const int N = 128;
```

```
_global_ void f(int *dev_a) {  
    unsigned int tid = threadIdx.x;
```

```
    if(tid < N) {  
        dev_a[tid] = tid * tid;  
    }  
}
```

```
int main(void) {
```

```
    int host_a[N];
```

```
    int *dev_a;
```

```
    cudaMalloc((void**)&dev_a, N * sizeof(int));
```

```
    for(int i = 0 ; i < N ; i++) {
```

```
        host_a[i] = i*i;
```

```
    }
```

```
    cudaMemcpy(dev_a, host_a, N * sizeof(int), cudaMemcpyHostToDevice);
```

```
    f<<<1, N>>>>(dev_a);
```



```
cudaMemcpy(host_a, dev_a, N * sizeof(int), cudaMemcpyDeviceToHost);
```

```
for(int i = 0 ; i < N ; i++) {  
    printf("%d ", host_a[i]);  
}  
}
```

Terminal OUTPUT:

```
#include <iostream>  
#include <numeric>  
#include <stdlib.h>  
#include <cuda.h>  
  
const int N = 128;  
  
__global__ void f(int *dev_a) {  
    unsigned int tid = threadIdx.x;  
  
    if(tid < N) {  
        dev_a[tid] = tid * tid;  
    }  
}  
  
int main(void) {  
    int host_a[N];  
    int *dev_a;  
    cudaMalloc((void**)&dev_a, N * sizeof(int));  
    for(int i = 0 ; i < N ; i++) {  
        host_a[i] = i*i;  
    }  
    cudaMemcpy(dev_a, host_a, N * sizeof(int), cudaMemcpyHostToDevice);  
    f<<1, N>>>(dev_a);  
  
    cudaMemcpy(host_a, dev_a, N * sizeof(int), cudaMemcpyDeviceToHost);  
  
    for(int i = 0 ; i < N ; i++) {  
        printf("%d ", host_a[i]);  
    }  
}
```

0 1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361 400 441 484 529 576 625 676 729 784 841 900 961 1024 1089 1156 1225 1296 1369 1444 1521 1600 1681 1764 1849 1936 2025 2116 2209 2304 2401 2500 2601 2704 2809 2916 3025 3136 3249 3364 3481 3600 3721 3844 3969 4096 4225 4356 4489 4624 4761 4900 5041 5184 5329 5476 5625 5776 5929 6084 6241 6400 6561 6724 6889 7056 7225 7396 7569 7744 7921 8100 8281 8464 8649 8836 9025 9216 9409 9604 9801 10000 10201 10404 10609 10816 11025 11236 11449 11664 11881 12100 12321 12544 12769 12996 13225 13456 13689 13924 14161 14400 14641 14884 15129 15376 15625 15876 16129 16384 16641 16900 17161 17424 17689 17956 18225 18496 18769 19044 19321 19600 19881 20164 20449 20736 21025 21316 21609 21904 22201 22500 22801 23104 23409 23716 24025 24336 24649 24964 25281 25600 25921 26244 26569 26896 27225 27556 27889 28224 28561 28900 29241 29584 29929 30276 30625 30976 31329 31684 32041 32400 32761 33124 33489 33856 34225 34596 34969 35344 35721 36100 36481 36864 37249 37636 38025 38416 38809 39204 39601 40000 40401 40804 41209 41616 42025 42436 42849 43264 43681 44100 44521 44944 45369 45796 46225 46656 47089 47524 47961 48400 48841 49284 49729 50176 50625 51076 51529 51984 52441 52900 53361 53824 54289 54756 55225 55696 56169 56644 57121 57600 58081 58564 59049 59536 60025 60516 61009 61504 62001 62500 63001 63504 64009 64516 65025 65536 66049 66564 67081 67600 68121 68644 69169 69696 70225 70756 71289 71824 72361 72900 73441 73984 74529 75076 75625 76176 76729 77284 77841 78400 78961 79524 80089 80656 81225 81796 82369 82944 83521 84100 84681 85264 85849 86436 87025 87616 88209 88804 89401 90000 90601 91204 91809 92416 93025 93636 94249 94864 95481 96100 96721 97344 97969 98596 99225 100000