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(π) | - |

**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-](https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit#prepare) [started-jetson-nano-2gb-devkit#prepare](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:



**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

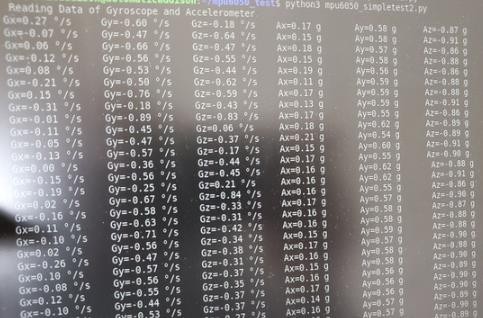
Gx **=** gyro\_x**/**131.0 Gy **=** gyro\_y**/**131.0 Gz **=** gyro\_z**/**131.0

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:



**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:

