TAP DETECTION PROGRAMMING:

Initialize Sensor in a typical configuration:
1. Set accelerometer ODR (Register 0x50h in Bank 0) ACCEL_ODR = 15 for 500 Hz (ODR of 200Hz or 1kHz may also be used)
2. Set power modes and filter configurations as shown below
   - For ODR up to 500Hz, set Accel to Low Power mode (Register 0x4Eh in Bank 0) ACCEL_MODE = 2 and ACCEL_LP_CLK_SEL = 0, (Register 0x4Dh in Bank 0) for low power mode Set filter settings as follows: ACCEL_DEC2_M2_ORD = 2 (Register 0x53h in Bank 0); ACCEL_UI_FILT_BW = 4 (Register 0x52h in Bank 0)
   - For ODR of 1kHz, set Accel to Low Noise mode (Register 0x4Eh in Bank 0) ACCEL_MODE = 1 Set filter settings as follows: ACCEL_UI_FILT_ORD = 2 (Register 0x53h in Bank 0); ACCEL_UI_FILT_BW = 0 (Register 0x52h in Bank 0)

 # [In the following code, we have used 500 Hz]

3. Wait 1 millisecond

Initialize APEX hardware:
1. Set TAP_TMAX to 2 (Register 0x47h in Bank 4)
2. Set TAP_TMIN to 3 (Register 0x47h in Bank 4)
3. Set TAP_TAVG to 3 (Register 0x47h in Bank 4)
4. Set TAP_MIN_JERK_THR to 17 (Register 0x46h in Bank 4)
5. Set TAP_MAX_PEAK_TOL to 2 (Register 0x46h in Bank 4)

 # [In the following code, we have used threshold as 63 and tolerance as 2]
--> Heat Map for variation of thres, tol and tmin --> thres and tol map.xlsx

6. Wait 1 millisecond
7. Enable TAP source for INT1 by setting bit 0 in register INT_SOURCE6 (Register 0x4Dh in Bank 4) to 1. Or if INT2 is selected for TAP, enable TAP source by setting bit 0 in register INT_SOURCE7 (Register 0x4Eh in Bank 4) to 1.
 # [In the following code, we went with INT1]
8. Wait 50 milliseconds
9. Turn on TAP feature by setting TAP_ENABLE to 1 (Register 0x56h in Bank 0)

Output registers:
1. Read interrupt register (Register 0x38h in Bank 0) for TAP_DET_INT
2. Read the tap count in TAP_NUM (Register 0x35h in Bank 0)
3. Read the tap axis in TAP_AXIS (Register 0x35h in Bank 0)
4. Read the polarity of tap pulse in TAP_DIR (Register 0x35h in Bank 0)

 # [For us, only 0x38 is needed]

Writing to registers:

Here, we use two different functions to write in a register:

- Reg_dir_write(add, val) – This function is used to directly write the value (val) into the given address (add). **This is used when the entire 8 bits of a register is to be written by us and there are no reserved bits.

```
void Reg_dir_write(unsigned long add, uint8_t val){  // write data to whole byte
Wire.beginTransmission(ICM);
Wire.write(add);                    // directs to the address add
Wire.write(val);                    // assigns value val to address add
Wire.endTransmission();
}
```

- Reg_write(add, and, or) – This function is used to write a specific value into some of the specific bits without affecting the values in the other bits. **This is used when all the 8 bits are not available but has some bits reserved.

```
void Reg_write(unsigned long add, uint8_t and_=255, uint8_t or_=0){
// write data to specific bits
Wire.beginTransmission(ICM);
Wire.write(add);
Wire.endTransmission();
Wire.requestFrom(ICM, 1);
if (Wire.available()){
mode=(Wire.read()&and_)|or_;
Wire.beginTransmission(ICM);
Wire.write(add);
Wire.write(mode);
Wire.endTransmission();}
}
```

We perform an AND operation with 1 at every place except where you wanna have 0. After the AND operation we will get a result where we will have 0 on the required location of the byte and then we perform an OR operation such that everyplace is kept 0 except where you want the value 1.

For example,

Let's take a random byte down below

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	1	0	1	0	1	0	0

Suppose we wanna assign value 0 on bit 4 and value 1 on bit 3 without affecting values in other bits.

Operation	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value
Initial	1	1	0	1	0	1	0	0	212
And	1	1	1	0	1	1	1	1	239
Result	1	1	0	0	0	1	0	0	196
Or	0	0	0	0	1	0	0	0	8
Final	1	1	0	0	1	1	0	0	204

Thus for getting 11001100 from 11010100, we need and & or value of 11101111 and 00001000.

TAP DETECTION CODE (SENDER – WEARABLE DEVICE):

```
#include <Wire.h>
#include <esp_now.h>
#include <WiFi.h>

const int ICM = 0x68;
int mode;
int flag = 0;
int count = 0;
int detect;

uint8_t rear_add[] = {0xA0,0xB7,0x65,0x61,0xCE,0xFC};

const uint8_t threshold = 254;  //threshold and tolerance variable --
252

//Structure example to receive data
//Must match the sender structure
//typedef struct rear_msg {
//  int rpm;
//  int duty;
//} rear_msg;

//Create a struct_message called msg
//rear_msg msg;

typedef struct tap_msg {
```

```
  //int id;
  //int throttle;
  int tap;
  //int mode;
} tap_msg;

tap_msg tmsg;

esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{
  Serial.print("\r\nLast Packet Send Status: \t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

//void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int
len) {
//  memcpy(&msg, incomingData, sizeof(msg));
  //Serial.print("Bytes received: ");
  //Serial.println(len);
  //Serial.print("rpm: ");
  //Serial.println(msg.rpm);
  //Serial.print("duty: ");
  //Serial.println(msg.duty);
  //Serial.println();}

int Reg_read(unsigned long add){     // reading data from the
particular register
  Wire.beginTransmission(ICM);
  Wire.write(add);
  Wire.endTransmission();
  Wire.requestFrom(ICM, 1);
  if (Wire.available()){
    return Wire.read();}

}

void Reg_write(unsigned long add, uint8_t and_=255, uint8_t or_=0){
// write data to specific bits
  Wire.beginTransmission(ICM);
  Wire.write(add);
```

```
  Wire.endTransmission();
  Wire.requestFrom(ICM, 1);
  if (Wire.available()){
    mode=(Wire.read()&and_)|or_;
    Wire.beginTransmission(ICM);
    Wire.write(add);
    Wire.write(mode);
    Wire.endTransmission();
  }
}

void Reg_dir_write(unsigned long add, uint8_t val){      // write data
to whole byte
  Wire.beginTransmission(ICM);
  Wire.write(add);
  Wire.write(val);
  Wire.endTransmission();
}

void Tap_detect(){     // ISR
  flag=1;
}

void setup() {

  Serial.begin(115200);

  // while (Serial.available() == 0) {
  // }

  // threshold = Serial.parseInt();
  // Serial.println(threshold);

  // while (Serial.available() == 0) {
  // }

  // uint8_t Tmin = Serial.parseInt();
  // Serial.println(Tmin);

  Wire.begin();

  Reg_dir_write(0x76, 0);    // set bank =0
```

```
  Reg_dir_write(0x50, 15);  //  ACCEL_ODR = 15 (500 Hz)

  Reg_write(0x4E, 254, 2);  // accel mode =2
  Reg_write(0x4D, 247);  // accel clk = 0
  Reg_write(0x53, 253, 4);  // ACCEL_DEC2_M2_ORD = 2
  Reg_write(0x52, 15, 64);  // ACCEL_UI_FILT_BW = 4

  delay(500);

  Reg_dir_write(0x76, 4);   // set bank =4

  Reg_dir_write(0x47, 93);  // TAP_TMAX to 2, TAP_TMIN to 3, TAP_TAVG
to 3
  Reg_dir_write(0x46, threshold);  // TAP_MIN_JERK_THR to 63,
TAP_MAX_PEAK_TOL to 2

  delay(500);

  Reg_write(0x4D, 255, 1);  // enable INT1 = 1

  delay(500);

  Reg_dir_write(0x76, 0);   // set bank =0

  Reg_write(0x56, 255, 64);  // enable INT1 = 1

  attachInterrupt(digitalPinToInterrupt(5), Tap_detect, RISING);    //
calling ISR at interrupt

  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info
  esp_now_register_send_cb(OnDataSent);

  // register peer
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  // register first peer
```

```
  memcpy(peerInfo.peer_addr, rear_add, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }

  //esp_now_register_recv_cb(OnDataRecv);

  delay(100);
  Serial.println("Setup Done");
  pinMode(LED_BUILTIN, OUTPUT);

}

void loop() {
  //tmsg.id = 0;
  tmsg.tap = 0;
  //tmsg.throttle = -1;
  //tmsg.mode = 0;

  if (flag==1){           // from ISR after detection
    detect = Reg_read(0x38);
    if (detect%2==1){                // double check register for tap
detection
      Serial.println("Tapp");
      flag = 0;
      count++;
      Serial.println(count);
      tmsg.tap = 1;
      digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the
voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the
voltage LOW
  delay(1000);
    }
  }
  esp_err_t result = esp_now_send(0, (uint8_t *) &tmsg, sizeof(tmsg ));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
```

```
    Serial.println("Error sending the data");
  }
  delay(20);

}
```

One of the issues which I found in this code is that the register 0x47 (APEX_CONFIG8) has a reserve value at bit 7. But here they used Reg_dir_write function to directly write the value (93) into it. But instead Reg_write should be used so that the reserved bit will be left untouched.
The and & or value should be 219 & 91 resp so that the output will be X1011011
Where, X is the reserved bit and TAP_TMAX = 2, TAP_TMIN = 3, TAP_TAVG = 3

Thus line117 should be `Reg_write(0x47, 219, 91);`

NOTE:

```
attachInterrupt(digitalPinToInterrupt(5), Tap_detect, RISING);
```

The interrupt pin number should be changed accordingly. In the wearable the interrupt pin of ICM is connected to the IO5 of ESP32.

Explanation:

```
Reg_dir_write(0x76, 0);   // set bank =0

  Reg_dir_write(0x50, 15);  //  ACCEL_ODR = 15 (500 Hz)

  Reg_write(0x4E, 254, 2);  // accel mode =2
  Reg_write(0x4D, 247);  // accel clk = 0
```

Here first the user bank is set to 0.

Then the ACCEL_ODR (0x50) value is set to 15 since for 500 Hz the binary value of 1111 should be written which is 15 in decimal.

In 0x4E (PWR_MGMT0), the 6 & 7 bit is reserved and 0 & 1 bit is for the ACCEL_MODE. We want the ACCEL_MODE to be 10 (decimal = 2) which places the accelerometer in Low Power (LP) Mode. Thus the and & or value should be 11111110 (decimal = 254) and 00000010 (decimal = 2) so that the output will be xxxxxx10 where x is the untouched bits.

In 0x4D, 7:4 bits are reserved and we need to set the bit 3 to 0 so that the Accelerometer LP mode uses the Wake Up oscillator clock. So the and value should be 11110111 (decimal = 247) and the or value should be 0 (default). So that the bit 3 will be 0 and rest will be left untouched.

Similarly the other registers are written with specific and & or values using Reg_write() function. Or if we don’t have any reserved bits we can directly use the Reg_dir_write() function.