LAMPIRAN

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>


// Pin definisi
const int lm393Pin = 2;   // Pin input dari LM393 atau sensor infrared
const int buzzerPin = 6;  // Pin output untuk buzzer
const int rpmThreshold1 = 245;  // Batas pertama RPM untuk menyalakan buzzer
const int rpmThreshold2 = 492;  // Batas kedua RPM untuk menyalakan buzzer
const int rpmThreshold3 = 734;  // Batas ketiga RPM untuk menyalakan buzzer

// Inisialisasi LCD dengan alamat I2C 0x27
LiquidCrystal_I2C lcd(0x27, 16, 2);

volatile int rpmCount = 0;
unsigned long lastTime = 0;
int lastRpm = 0; // Variabel untuk menyimpan RPM terakhir

// Variabel untuk menyimpan beberapa pembacaan RPM untuk rata-rata
const int numReadings = 10;
int readings[numReadings]; // Array untuk menyimpan pembacaan
int readIndex = 0; // Index pembacaan saat ini
int total = 0; // Jumlah total pembacaan
int averageRpm = 0; // Rata-rata RPM

// Status untuk melacak apakah buzzer sudah berbunyi
bool hasBuzzer1Triggered = false;
bool hasBuzzer2Triggered = false;
bool hasBuzzer3Triggered = false;

void setup() {
  // Inisialisasi LCD
  lcd.init();
  lcd.backlight();
  lcd.print("RPM: 0");

  // Inisialisasi pin
  pinMode(lm393Pin, INPUT_PULLUP); // Menggunakan internal pull-up jika diperlukan
  pinMode(buzzerPin, OUTPUT);

  // Inisialisasi interrupt
  attachInterrupt(digitalPinToInterrupt(lm393Pin), rpmCounter, FALLING);

  // Inisialisasi array pembacaan
  for (int i = 0; i < numReadings; i++) {
    readings[i] = 0;
  }
```

```
  // Debug: Konfirmasi bahwa interrupt berfungsi
  lcd.setCursor(0, 1);
  lcd.print("IR Ready");
}

void loop() {
  // Hitung RPM
  unsigned long currentTime = millis();
  unsigned long elapsedTime = currentTime - lastTime;

  if (elapsedTime >= 1000) { // 1 detik
    lastTime = currentTime;
    noInterrupts(); // Nonaktifkan interrupt sementara
    int rpm = (rpmCount * 60) / 2; // Menghitung RPM
    rpmCount = 0; // Reset counter
    interrupts(); // Aktifkan kembali interrupt

    // Stabilkan RPM dengan menjaga nilai terakhir jika tidak ada perubahan
    if (rpm != lastRpm) {
      lastRpm = rpm;

      // Update rata-rata pembacaan RPM
      total -= readings[readIndex];       // Kurangi pembacaan terlama dari total
      readings[readIndex] = rpm;          // Ganti dengan pembacaan baru
      total += readings[readIndex];       // Tambahkan pembacaan baru ke total
      readIndex = (readIndex + 1) % numReadings; // Pindah ke index berikutnya

      averageRpm = total / numReadings;   // Hitung rata-rata RPM

      // Tampilkan pada LCD
      lcd.clear();
      lcd.print("RPM: ");
      lcd.print(averageRpm);

      // Alarm berdasarkan RPM rata-rata
      if (averageRpm > rpmThreshold3 && !hasBuzzer3Triggered) {
        beepBuzzer(1, 4000); // Beep terus menerus selama 4 detik
        hasBuzzer3Triggered = true;
      } else if (averageRpm > rpmThreshold2 && !hasBuzzer2Triggered) {
        beepBuzzer(2, 2000); // 2 kali beep selama 2 detik
        hasBuzzer2Triggered = true;
      } else if (averageRpm > rpmThreshold1 && !hasBuzzer1Triggered) {
        beepBuzzer(1, 2000); // 1 kali beep selama 2 detik
        hasBuzzer1Triggered = true;
      }

      // Cek jika semua threshold sudah tercapai
      if (hasBuzzer1Triggered && hasBuzzer2Triggered && hasBuzzer3Triggered) {
        resetSystem();
```

```
    }
   }
  }
}

void rpmCounter() {
  rpmCount++;
}

// Fungsi untuk membunyikan buzzer dengan jumlah beep tertentu
void beepBuzzer(int times, int duration) {
  int beepDuration = duration / times; // Durasi setiap beep
  for (int i = 0; i < times; i++) {
    digitalWrite(buzzerPin, HIGH);
    delay(beepDuration / 2); // Waktu bunyi
    digitalWrite(buzzerPin, LOW);
    delay(beepDuration / 2); // Waktu jeda
  }
}

// Fungsi untuk mereset sistem setelah semua threshold tercapai
void resetSystem() {
  hasBuzzer1Triggered = false;
  hasBuzzer2Triggered = false;
  hasBuzzer3Triggered = false;
  lcd.clear();
  lcd.print("System Reset");
  delay(2000); // Tunda selama 2 detik sebelum memulai kembali
  lcd.clear();
}
```