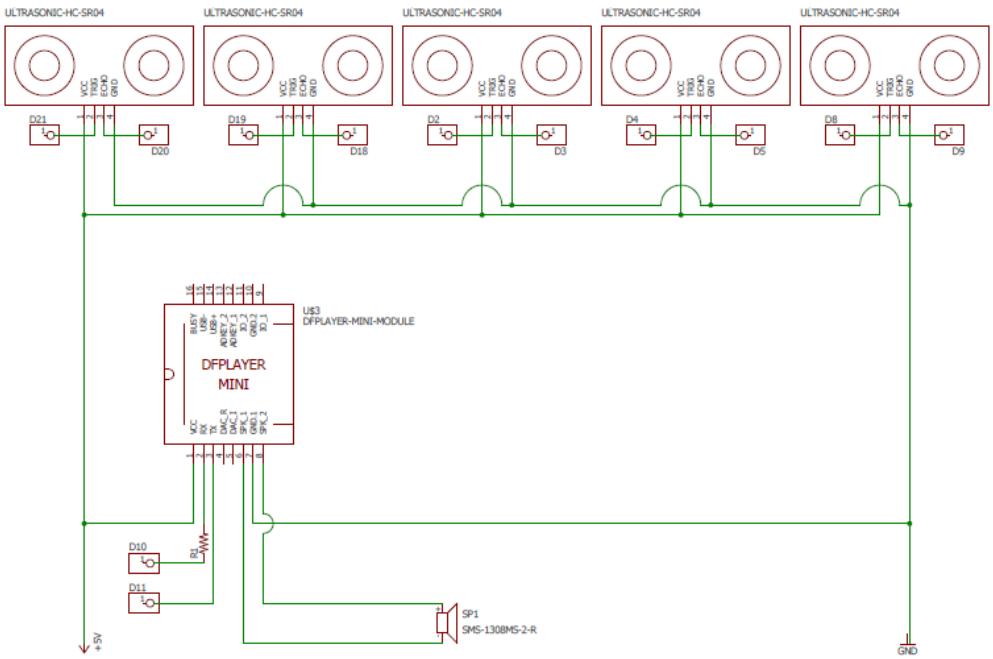
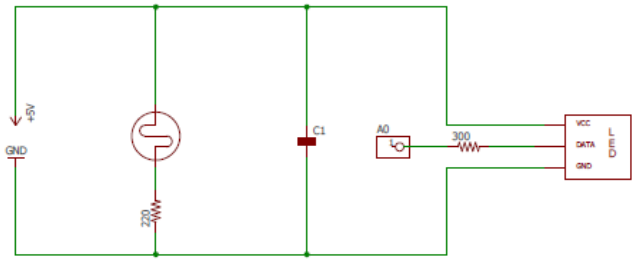
LAMPIRAN A

SKEMA RANGKAIAN





LAMPIRAN B

PROGRAM

```
#include "Arduino.h"
#include "SoftwareSerial.h"
#include "DFRobotDFPlayerMini.h"
#include "Sensor.h"
#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
  #include <avr/power.h>
#endif
#define PIN 6
#define NUM_LEDS 60

SoftwareSerial mySoftwareSerial(12, 13); // RX, TX

DFRobotDFPlayerMini myDFPlayer;

void printDetail(uint8_t type, int value);

Sensor * s1;
Sensor * s2;
Sensor * s3;
```

```
Sensor * s4;
Sensor * s5;

void setup() {
  mySoftwareSerial.begin(9600);
  strip.begin();
  strip.show(); // Mematikan seluruh led

  Serial.begin(115200);

  s1 = new Sensor(21,20);
  s2 = new Sensor(19,18);
  s3 = new Sensor(2,3);
  s4 = new Sensor(4,5);
  s5 = new Sensor(10,11);

  Serial.println();
  Serial.println(F("Initializing DFPlayer ... (May take 3~5 seconds)"));

  if (!myDFPlayer.begin(mySoftwareSerial)) {  // SoftwareSerial untuk berinteraksi dengan mp3
    Serial.println(F("Unable to begin:"));
    Serial.println(F("1.Please recheck the connection!"));
```

```
    Serial.println(F("2.Please insert the SD card!"));
    while(true);
  }


  Serial.println(F("DFPlayer Mini online."));

  myDFPlayer.volume(18);  //Set volume value. From 0 to 30
}
const byte LDRSensor = 0;

bool checkLDR ()
  {
  int LDRValue = analogRead(LDRSensor);
  return LDRValue > 100;
  }

void loop() {
  colorWipe(strip.Color(255, 0, 0), 50, checkLDR);
  colorWipe(strip.Color(0, 255, 0), 50, checkLDR);
  colorWipe(strip.Color(0, 0, 255), 50, checkLDR);
  colorWipe(strip.Color(0, 0, 0, 255), 50);
  theaterChase(strip.Color(127, 127, 127), 50, checkLDR);
```

```
theaterChase(strip.Color(127, 0, 0), 50, checkLDR);
theaterChase(strip.Color(0, 0, 127), 50, checkLDR);
rainbow(20, checkLDR);
rainbowCycle(20, checkLDR);
theaterChaseRainbow(50, checkLDR);
RGBLoop(20 ,checkLDR);
FadeInOut(0x45, 0xFF, 0xDA, 20, checkLDR);
FadeInOut(0xC7, 0x2E, 0xFF, 20, checkLDR);
FadeInOut(0xFF, 0x8D, 0x29, 20, checkLDR);
FadeInOut(0x8F, 0xFF, 0xF8, 20, checkLDR);
CylonBounce(0xff, 0, 0, 4, 10, 50, 20, checkLDR);
TwinkleRandom(20, 100, false, 20, checkLDR);
Fire(55,120,15, 100, checkLDR);

 // perhitungan jarak
 unsigned long dist1 = s1->dist();
 unsigned long dist2 = s2->dist();
 unsigned long dist3 = s3->dist();
 unsigned long dist4 = s4->dist();
 unsigned long dist5 = s5->dist();

if (dist1 < 10 && dist1 > 1){
  Serial.print("Tone ");
```

```
  Serial.print("1, ");
  Serial.print("Proximity -- ");
  Serial.println(dist1);
  myDFPlayer.playMp3Folder(1);
  delay(800);
 }

if (dist2 < 10 && dist2 > 1) {
  Serial.print("Tone ");
  Serial.print("2. ");
  Serial.print("Proximity -- ");
  Serial.println(dist2);
  myDFPlayer.playMp3Folder(2);
  delay(800);
 }

  if (dist3 < 10 && dist3 >1) {
  Serial.print("Tone ");
  Serial.print("3, ");
  Serial.print("Proximity -- ");
  Serial.println(dist3);
  myDFPlayer.playMp3Folder(3);
  delay(800);
```

```
  }


   if (dist4 < 10 && dist4 > 1) {
   Serial.print("Tone ");
   Serial.print("4, ");
   Serial.print("Proximity -- ");
   Serial.println(dist4);
   myDFPlayer.playMp3Folder(4);
   delay(800);
  }


   if (dist5 < 10 && dist5 > 1) {
   Serial.print("Tone ");
   Serial.print("5, ");
   Serial.print("Proximity -- ");
   Serial.println(dist5);
   myDFPlayer.playMp3Folder(5);
   delay(800);
  }


  if (myDFPlayer.available()) {
   printDetail(myDFPlayer.readType(), myDFPlayer.read()); // Detail untuk
mengetahui adakah error atau malfungsi dari DFPlayer
```

```
  }
}

void printDetail(uint8_t type, int value){

  switch (type) {
    case TimeOut:
      Serial.println(F("Time Out!"));
      break;
    case WrongStack:
      Serial.println(F("Stack Wrong!"));
      break;
    case DFPlayerCardInserted:
      Serial.println(F("Card Inserted!"));
      break;
    case DFPlayerCardRemoved:
      Serial.println(F("Card Removed!"));
      break;
    case DFPlayerCardOnline:
      Serial.println(F("Card Online!"));
      break;
    case DFPlayerPlayFinished:
      Serial.print(F("Number:"));
```

```
  Serial.print(value);
  Serial.println(F(" Play Finished!"));
  break;
case DFPlayerError:
  Serial.print(F("DFPlayerError:"));
  switch (value) {
    case Busy:
      Serial.println(F("Card not found"));
      break;
    case Sleeping:
      Serial.println(F("Sleeping"));
      break;
    case SerialWrongStack:
      Serial.println(F("Get Wrong Stack"));
      break;
    case CheckSumNotMatch:
      Serial.println(F("Check Sum Not Match"));
      break;
    case FileIndexOut:
      Serial.println(F("File Index Out of Bound"));
      break;
    case FileMismatch:
      Serial.println(F("Cannot Find File"));
```

```
        break;
      case Advertise:
        Serial.println(F("In Advertise"));
        break;
      default:
        break;
    }
    break;
  default:
    break;
 }
}

// Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t wait, AbortFunction fAbort) {
 for(uint16_t i=0; i<strip.numPixels(); i++) {
  if (fAbort && fAbort ())
    return;
  strip.setPixelColor(i, c);
  strip.show();
  delay(wait);
 }
}
```

```
void rainbow(uint8_t wait, AbortFunction fAbort) {
  uint16_t i, j;

  for(j=0; j<256; j++) {
    for(i=0; i<strip.numPixels(); i++) {
      if (fAbort && fAbort ())
        return;
      strip.setPixelColor(i, Wheel((i+j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

// Slightly different, this makes the rainbow equally distributed throughout
void rainbowCycle(uint8_t wait, AbortFunction fAbort) {
  uint16_t i, j;

  for(j=0; j<256*5; j++) { // 5 cycles of all colors on wheel
    if (fAbort && fAbort ())
      return;
    for(i=0; i< strip.numPixels(); i++) {
```

```
      strip.setPixelColor(i, Wheel(((i * 256 / strip.numPixels()) + j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

//Theatre-style crawling lights.AbortFunction fAbort
void theaterChase(uint32_t c, uint8_t wait, AbortFunction fAbort) {
  for (int j=0; j<10; j++) {  //do 10 cycles of chasing
    for (int q=0; q < 3; q++) {
      if (fAbort && fAbort ())
        return;
      for (uint16_t i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, c);    //turn every third pixel on
      }
      strip.show();

      delay(wait);

      for (uint16_t i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, 0);        //turn every third pixel off
      }
```

```
    }
  }
}

//Theatre-style crawling lights with rainbow effect
void theaterChaseRainbow(uint8_t wait, AbortFunction fAbort) {
  for (int j=0; j < 256; j++) {     // cycle all 256 colors in the wheel
    for (int q=0; q < 3; q++) {
      if (fAbort && fAbort ())
        return;
      for (uint16_t i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, Wheel( (i+j) % 255));    //turn every third pixel on
      }
      strip.show();

      delay(wait);

      for (uint16_t i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, 0);        //turn every third pixel off
      }
    }
  }
}
```

```
void RGBLoop(uint8_t wait, AbortFunction fAbort) {
  for(int j = 0; j < 3; j++ ) {
    // Fade IN
    for(int k = 0; k < 256; k++) {
      if (fAbort && fAbort ())
       return;
         switch(j) {
       case 0: setAll(k,0,0); break;
       case 1: setAll(0,k,0); break;
       case 2: setAll(0,0,k); break;
      }
     showStrip();
     delay(3);
    }
    // Fade OUT
    for(int k = 255; k >= 0; k--) {
     switch(j) {
       case 0: setAll(k,0,0); break;
       case 1: setAll(0,k,0); break;
       case 2: setAll(0,0,k); break;
      }
     showStrip();
```

```
      delay(3);
    }
  }
}



// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if(WheelPos < 85) {
    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  if(WheelPos < 170) {
    WheelPos -= 85;
    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}

void showStrip() {
```

```
  strip.show();
}

 void setPixel(int Pixel, byte red, byte green, byte blue) {
   strip.setPixelColor(Pixel, strip.Color(red, green, blue));
 }

 void setAll(byte red, byte green, byte blue) {
  for(int i = 0; i < NUM_LEDS; i++ ) {
    setPixel(i, red, green, blue);
  }
  showStrip();
}

void FadeInOut(byte red, byte green, byte blue, uint8_t wait, AbortFunction fAbort){
  float r, g, b;

  for(int k = 0; k < 256; k=k+1) {
    if (fAbort && fAbort ())
        return;
    r = (k/256.0)*red;
    g = (k/256.0)*green;
    b = (k/256.0)*blue;
```

```
  setAll(r,g,b);
  showStrip();
 }

 for(int k = 255; k >= 0; k=k-2) {
  r = (k/256.0)*red;
  g = (k/256.0)*green;
  b = (k/256.0)*blue;
  setAll(r,g,b);
  showStrip();
 }
}

void Strobe(byte red, byte green, byte blue, int StrobeCount, int FlashDelay, int EndPause, uint8_t wait, AbortFunction fAbort){
 for(int j = 0; j < StrobeCount; j++) {

  if (fAbort && fAbort ())
      return;
  setAll(red,green,blue);
  showStrip();
  delay(FlashDelay);
  setAll(0,0,0);
```

```
   showStrip();
   delay(FlashDelay);
  }

 delay(EndPause);
}

void CylonBounce(byte red, byte green, byte blue, int EyeSize, int SpeedDelay, int ReturnDelay, uint8_t wait, AbortFunction fAbort ){

  for(int i = 0; i < NUM_LEDS-EyeSize-2; i++) {
   if (fAbort && fAbort ())
       return;
   setAll(0,0,0);
   setPixel(i, red/10, green/10, blue/10);
   for(int j = 1; j <= EyeSize; j++) {
     setPixel(i+j, red, green, blue);
   }
   setPixel(i+EyeSize+1, red/10, green/10, blue/10);
   showStrip();
   delay(SpeedDelay);
  }
```

```
  delay(ReturnDelay);

  for(int i = NUM_LEDS-EyeSize-2; i > 0; i--) {
   setAll(0,0,0);
   setPixel(i, red/10, green/10, blue/10);
   for(int j = 1; j <= EyeSize; j++) {
     setPixel(i+j, red, green, blue);
   }
   setPixel(i+EyeSize+1, red/10, green/10, blue/10);
   showStrip();
   delay(SpeedDelay);
  }

  delay(ReturnDelay);
}

void TwinkleRandom(int Count, int SpeedDelay, boolean OnlyOne, uint8_t wait, AbortFunction fAbort ) {
  setAll(0,0,0);

  for (int i=0; i<Count; i++) {
     if (fAbort && fAbort ())
        return;
```

```
    setPixel(random(NUM_LEDS),random(0,255),random(0,255),random(0,255));
    showStrip();
    delay(SpeedDelay);
    if(OnlyOne) {
      setAll(0,0,0);
    }
   }

  delay(SpeedDelay);
}

void Fire(int Cooling, int Sparking, int SpeedDelay, uint8_t wait, AbortFunction fAbort ) {
  static byte heat[NUM_LEDS];
  int cooldown;

  // Step 1.  Cool down every cell a little
  for( int i = 0; i < NUM_LEDS; i++) {
        if (fAbort && fAbort ())
       return;
    cooldown = random(0, ((Cooling * 10) / NUM_LEDS) + 2);

    if(cooldown>heat[i]) {
```

```
    heat[i]=0;
  } else {
    heat[i]=heat[i]-cooldown;
  }
}

// Step 2.  Heat from each cell drifts 'up' and diffuses a little
for( int k= NUM_LEDS - 1; k >= 2; k--) {
  heat[k] = (heat[k - 1] + heat[k - 2] + heat[k - 2]) / 3;
}

// Step 3.  Randomly ignite new 'sparks' near the bottom
if( random(255) < Sparking ) {
  int y = random(7);
  heat[y] = heat[y] + random(160,255);
  //heat[y] = random(160,255);
}

// Step 4.  Convert heat to LED colors
for( int j = 0; j < NUM_LEDS; j++) {
  setPixelHeatColor(j, heat[j] );
}
```

```
  showStrip();
  delay(SpeedDelay);
}

void setPixelHeatColor (int Pixel, byte temperature) {
  // Scale 'heat' down from 0-255 to 0-191
  byte t192 = round((temperature/255.0)*191);

  // calculate ramp up from
  byte heatramp = t192 & 0x3F; // 0..63
  heatramp <<= 2; // scale up to 0..252

  // figure out which third of the spectrum we're in:
  if( t192 > 0x80) {                    // hottest
    setPixel(Pixel, 255, 255, heatramp);
  } else if( t192 > 0x40 ) {            // middle
    setPixel(Pixel, 255, heatramp, 0);
  } else {                              // coolest
    setPixel(Pixel, heatramp, 0, 0);
  }
}
```

```
void BouncingColoredBalls(int BallCount, byte colors[][3], uint8_t wait, AbortFunction fAbort) {
  float Gravity = -9.81;
  int StartHeight = 1;

  float Height[BallCount];
  float ImpactVelocityStart = sqrt( -2 * Gravity * StartHeight );
  float ImpactVelocity[BallCount];
  float TimeSinceLastBounce[BallCount];
  int   Position[BallCount];
  long  ClockTimeSinceLastBounce[BallCount];
  float Dampening[BallCount];

  for (int i = 0 ; i < BallCount ; i++) {
   if (fAbort && fAbort ())
      return;
   ClockTimeSinceLastBounce[i] = millis();
   Height[i] = StartHeight;
   Position[i] = 0;
   ImpactVelocity[i] = ImpactVelocityStart;
   TimeSinceLastBounce[i] = 0;
   Dampening[i] = 0.90 - float(i)/pow(BallCount,2);
  }
```

```
 while (true) {
  for (int i = 0 ; i < BallCount ; i++) {
   TimeSinceLastBounce[i] =  millis() - ClockTimeSinceLastBounce[i];
   Height[i] = 0.5 * Gravity * pow( TimeSinceLastBounce[i]/1000 , 2.0 ) + ImpactVelocity[i] * TimeSinceLastBounce[i]/1000;

   if ( Height[i] < 0 ) {
    Height[i] = 0;
    ImpactVelocity[i] = Dampening[i] * ImpactVelocity[i];
    ClockTimeSinceLastBounce[i] = millis();

    if ( ImpactVelocity[i] < 0.01 ) {
     ImpactVelocity[i] = ImpactVelocityStart;
    }
   }
   Position[i] = round( Height[i] * (NUM_LEDS - 1) / StartHeight);
  }

  for (int i = 0 ; i < BallCount ; i++) {
   setPixel(Position[i],colors[i][0],colors[i][1],colors[i][2]);
  }
```

```
  showStrip();
  setAll(0,0,0);
}
```