

# **Step 1 :**

We take input of data from the arduino uno microcontroller using a pulse sensor. Pulse sensors have in-built noise cancellation and amplification ICS.

**Code of Arduino Uno Microcontroller to take input from serial**

/\*

Code to detect pulses from the PulseSensor,

using an interrupt service routine.

Here is a link to the tutorial\

https://pulsesensor.com/pages/getting-advanced

Copyright World Famous Electronics LLC - see LICENSE

Contributors:

Joel Murphy, https://pulsesensor.com

Yury Gitman, https://pulsesensor.com

Bradford Needham, @bneedhamia, https://bluepapertech.com

Licensed under the MIT License, a copy of which

should have been included with this software.

This software is not intended for medical use.

\*/

/\*

Every Sketch that uses the PulseSensor Playground must

define USE\_ARDUINO\_INTERRUPTS before including PulseSensorPlayground.h.

Here, #define USE\_ARDUINO\_INTERRUPTS true tells the library to use

interrupts to automatically read and process PulseSensor data.

See ProcessEverySample.ino for an example of not using interrupts.

\*/

#define USE\_ARDUINO\_INTERRUPTS true

#include <PulseSensorPlayground.h>

/\*

The format of our output.

Set this to PROCESSING\_VISUALIZER if you're going to run

the Processing Visualizer Sketch.

See https://github.com/WorldFamousElectronics/PulseSensor\_Amped\_Processing\_Visualizer

Set this to SERIAL\_PLOTTER if you're going to run

the Arduino IDE's Serial Plotter.

\*/

//const int OUTPUT\_TYPE = SERIAL\_PLOTTER;

const int OUTPUT\_TYPE = PROCESSING\_VISUALIZER;

/\*

Pinout:

PULSE\_INPUT = Analog Input. Connected to the pulse sensor

purple (signal) wire.

PULSE\_BLINK = digital Output. Connected to an LED (and 220 ohm resistor)

that will flash on each detected pulse.

PULSE\_FADE = digital Output. PWM pin onnected to an LED (and resistor)

that will smoothly fade with each pulse.

NOTE: PULSE\_FADE must be a pin that supports PWM. Do not use

pin 9 or 10, because those pins' PWM interferes with the sample timer.

\*/

const int PULSE\_INPUT = A0;

const int PULSE\_BLINK = 13; // Pin 13 is the on-board LED

const int PULSE\_FADE = 5;

const int THRESHOLD = 550; // Adjust this number to avoid noise when idle

/\*

All the PulseSensor Playground functions.

\*/

PulseSensorPlayground pulseSensor;

void setup() {

/\*

Use 115200 baud because that's what the Processing Sketch expects to read,

and because that speed provides about 11 bytes per millisecond.

If we used a slower baud rate, we'd likely write bytes faster than

they can be transmitted, which would mess up the timing

of readSensor() calls, which would make the pulse measurement

not work properly.

\*/

Serial.begin(115200);

// Configure the PulseSensor manager.

pulseSensor.analogInput(PULSE\_INPUT);

pulseSensor.blinkOnPulse(PULSE\_BLINK);

pulseSensor.fadeOnPulse(PULSE\_FADE);

pulseSensor.setSerial(Serial);

pulseSensor.setOutputType(OUTPUT\_TYPE);

pulseSensor.setThreshold(THRESHOLD);

// Now that everything is ready, start reading the PulseSensor signal.

if (!pulseSensor.begin()) {

/\*

PulseSensor initialization failed,

likely because our particular Arduino platform interrupts

aren't supported yet.

If your Sketch hangs here, try PulseSensor\_BPM\_Alternative.ino,

which doesn't use interrupts.

\*/

for(;;) {

// Flash the led to show things didn't work.

digitalWrite(PULSE\_BLINK, LOW);

delay(50);

digitalWrite(PULSE\_BLINK, HIGH);

delay(50);

}

}

}

void loop() {

/\*

Wait a bit.

We don't output every sample, because our baud rate

won't support that much I/O.

\*/

delay(20);

// write the latest sample to Serial.

pulseSensor.outputSample();

/\*

If a beat has happened since we last checked,

write the per-beat information to Serial.

\*/

if (pulseSensor.sawStartOfBeat()) {

pulseSensor.outputBeat();

}

}