Investigation of the accuracy of the Arduino Mega 2560 internal clock and the Adafruit Ultimate GPS breakout board v3 PPS signal.

Yuqing Zhai

Introduction

In real-world experiment data collection, microcontrollers like Arduino are frequently used. In some situations, like measuring the acceleration of a high-speed object, or the period of a quickly oscillated pendulum, accuracy in measurement of time is required. This report investigates the accuracy and reliability of the internal clock of Arduino Mega 2560, one brand of Arduino. The report further develops a method of using the PPS signal from the GPS module to calibrate internal clocks and reduce the inaccuracy significantly. Finally, this report measures the inaccuracy of the PPS signal from the Adafruit Ultimate GPS breakout board.

Measurement of Accuracy of internal clock

From the official documentation [3], the Arduino internal clock could be accessed via `micros()` function, which gives an integer that indicates the time passed in microseconds since the program started (the timestamp). To investigate the accuracy of this function, we could use the PPS Signal from the GPS module.

The GPS module will send a PPS signal that lasts for 50-100ms [1] every second. We could log the result of micros () function every time we received the signal. For this report, we use the Arduino Mega 2560 and Adafruit Ultimate GPS breakout board v3. The Arduino Mega 2560 has only a 16MHz clock rate, and according to the documentation of micros () function, it only has resolution of 4us. [3] The GPS breakout board v3, on the other hand, "have typical accuracy ranging 10ns" [1]. Therefore, the influence of the inaccuracy of PPS signal is negligible.

The schematic and Fritzing graph of the circuit are shown below:

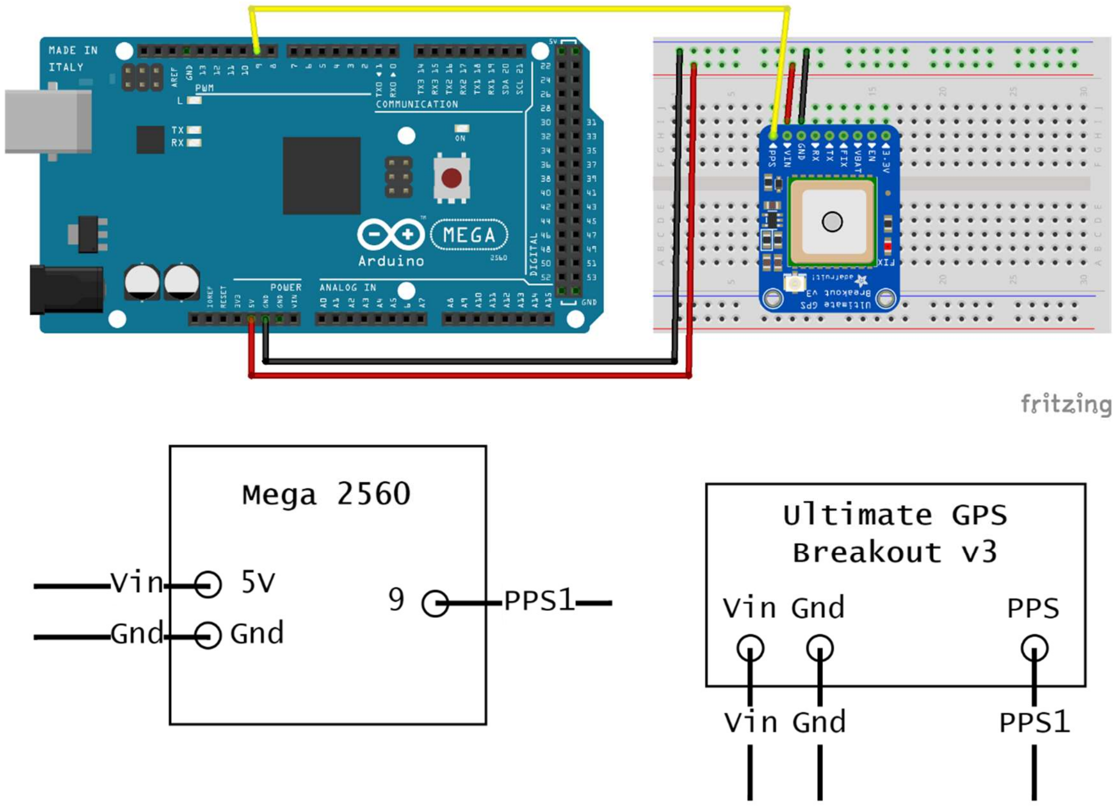


Fig. 1. The schematic and Fritzing graph for the accuracy-testing circuit

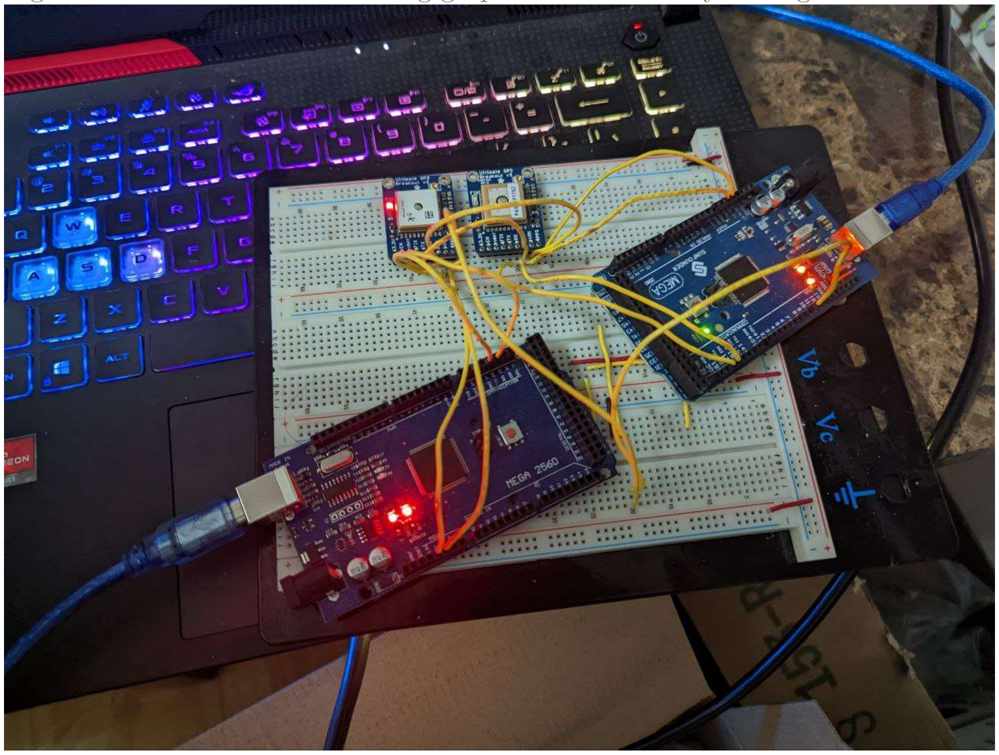


Fig. 2. The picture was taken for the real experiment setup. The schematic shown above was used for each Arduino Mega 2560 in the picture.

The data collection program will simply detect the PPS signal, and output the current timestamp to the Serial (code shown in Appendix I):

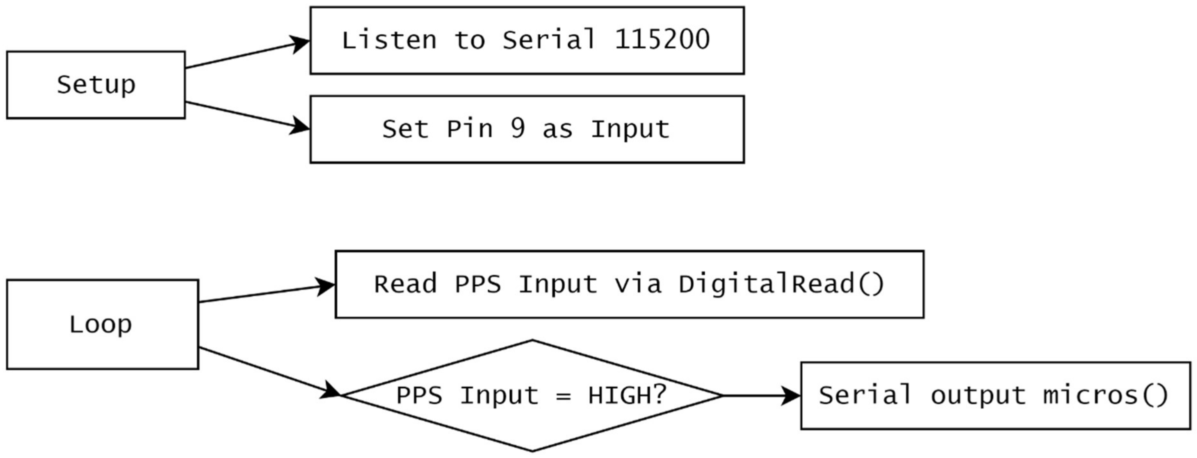


Fig. 3. The illustration of the accuracy testing program.

The time for the start of PPS signal is thus contained in the Serial output. We could get the inaccuracy in microseconds by:

$$\Delta t_{i,\ \mathrm{error}} = \sum_{i=1}^{n} \left(t_{i,\ \mathrm{PPS}} - t_{i-1,\,\mathrm{PPS}}\right) - 1000000$$

And its average and standard deviation:

$$\langle \Delta t_{err} \rangle = \frac{1}{n}\sum_{i=1}^{n} \Delta t_{i,\,\mathrm{error}} \qquad std(\Delta t) = \frac{1}{n}\left(\sum_{i=1}^{n} \left(\Delta t_{i,\,\mathrm{error}} - \Delta t\right)^2\right)^{\frac{1}{2}}$$

Notice, the delay caused by the calling function digitalRead() function could be ignored. It's expected that the programs will always delay the same amount as long as large amount of data is collected. Thus the Δt calculated will not be affected, as shown below:

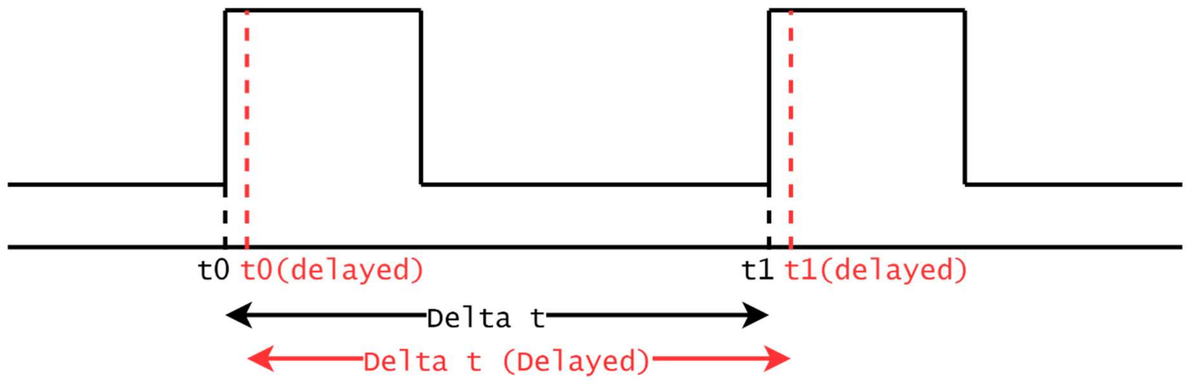


Fig. 4. The graph shows that delay in measurement won't cause the change in Delta t.

One simplest calibration method is to dilate the internal time return by micros() by a factor:

$$p_{\text{dilation}} = \frac{\langle \Delta t_{err} \rangle}{1000000} \cdot 100\%$$

$$t_{\text{calibrated}} = \frac{t_{\text{original}}}{p_{\text{dilation}}}$$

We set up a circuit to test this method. In the circuit, one Mega 2560 sends the signal, and the other Mega 2560 receives it. The signal moves at near-light speed so the time delay here could be ignored. The time that each signal sent and received is recorded and later calibrated using the previously collected data for every Arduino Mega 2560. We wanted to see how much we could sync these two Arduino Mega 2560 using the calibration. The schematic and fritzing graph is shown below. Mega 2560's p_{dilation} are measured by previous setup.

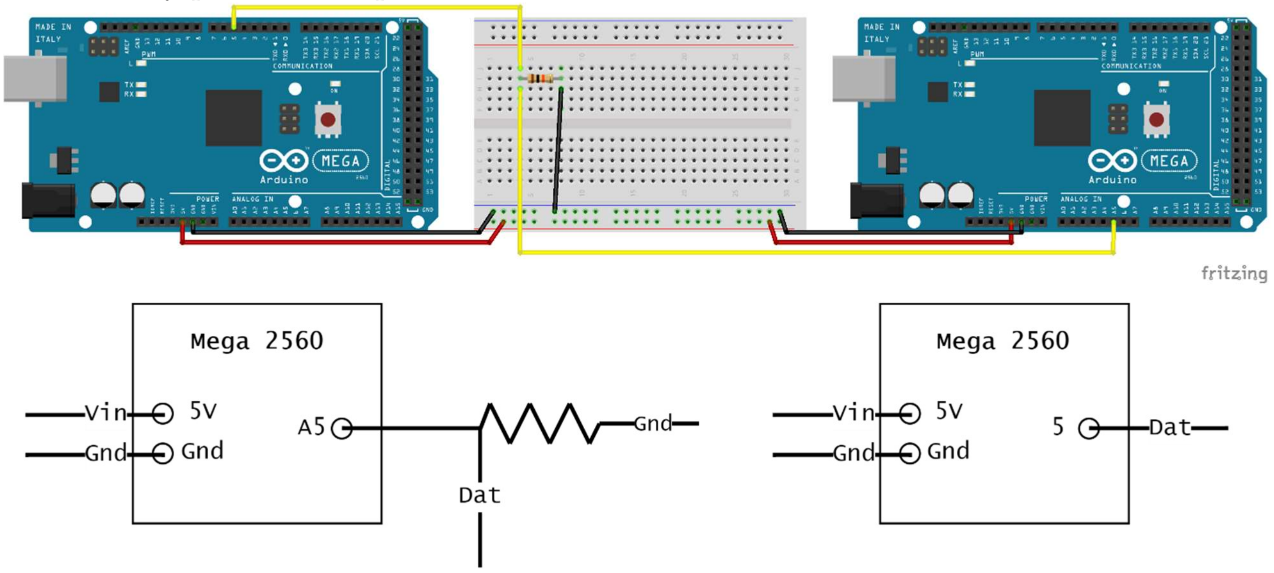


Fig. 5, the schematic and Fritzing graph for circuit used to check the effect of calibration

The data collection program, when either send or receive a signal, will record the time and its corresponding index. (Code in Appendix II) The difference in the time could then be calculated:

$$\Delta t_{\text{error after calibration}} = t_{\text{calibrated, first}} - t_{\text{calibrated, second}}$$

The result is analyzed in the final section.

Measurement of the Accuracy of PPS Signal

The PPS signal usually "has an accuracy ranging from 12 picoseconds to a few microseconds per second" [4]. For the Adafruit Ultimate GPS breakout board v3 its PPS signal "have typical accuracy ranging 10ns" [1]. However, as previously mentioned, the theoretical maximum accuracy is 4us [3] for the internal clock of Arduino Mega 2560. This means we are unable to measure any noticeable difference in the PPS signals using the internal clock directly. So, a different approach was used with the flipflop gate.

A flip-flop gate looks like below:

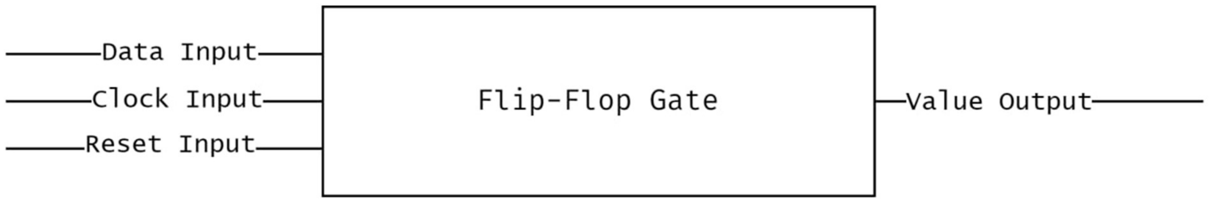


Fig. 6. This illustration of the flipflop gate.

When the flip-flop detects that Clock Input goes from LOW to HIGH, the gate will record the Data Input at that instant and hold it to Value Output. When Reset Input goes to HIGH, the Value Output will be reset to LOW. Notice, before the Clock Input goes from LOW to HIGH, the data input must already be set up to the desired value (HIGH or LOW) for some time in order to be recorded. This time is called "minimum setup time", as illustrated below:

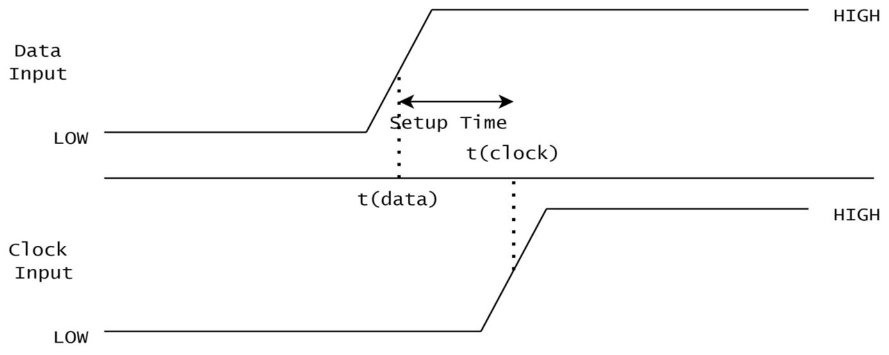


Fig. 7. The time when data arrives is indicated by t(data), and the time when clock input arrives is indicated by t(clock), the minimum time needed to successfully record the data is the setup time shown in the graph.

Therefore, we could prepare two GPS modules. We could then use the first one's PPS signal as the clock input, and the second one's PPS signal as the data input. If the second PPS signal arrives at least setup_time before the first PPS signal, the Flip-Flop gate could then successfully record the second PPS signal, and we could use Arduino Mega 2560 to read HIGH from the Value Output. Otherwise, the Flip-Flop gate could not record the second PPS signal, and we would get a LOW from the Value Output.

Without the loss of generosity, let's assume the clock input signal arrives earlier than the data input signal, then the Flip-Flop would not record the data input signal. (Show in figure below) We could gradually delay the clock input signal to the threshold where the flipflop gate starts to record the Data Input Signal and we start to get HIGH reading from the Value Output. As previously mentioned, this is the point where the Data Input signal is "Minimum Setup Time" earlier than the Clock Input signal.

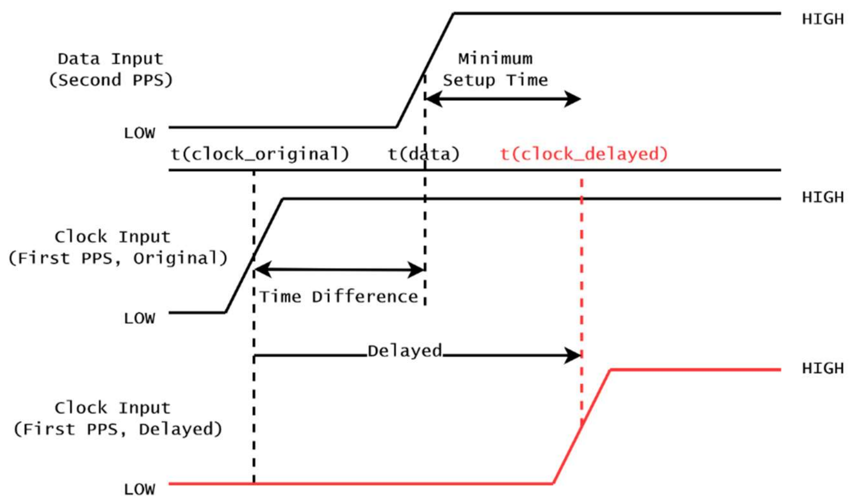


Fig. 8. The time when the clock input originally arrives is indicated by t(clock_original). The time when the delayed clock input comes is indicated by t(clock_delayed), other time are indicated on the graph.

From the graph, we could see that the Time Difference between the First PPS signal and Second PPS Signal is thus:

$$\Delta t = \Delta t(\text{Delayed}) - \Delta t(\text{Minimum Setup Time})$$

To measure the minimum setup time, we could use First PPS signal for Data Input and Clock Input, in this case, the $\Delta t = 0$ and the formula thus changed to

$$0 = \Delta t(\text{Delayed}) - \Delta t(\text{Minimum Setup Time})$$
$$\Delta t(\text{Delayed}) = \Delta t(\text{Minimum Setup Time})$$

In the experiment, we use long #22 wires to delay the signal, the time delayed could be calculate as below:

$$\Delta t(Delayed) = \frac{x_{\text{Wire Length}}}{v_{\text{Current Speed}}}$$

The $v_{\text{Current Speed}}$ is measured via the oscilloscope and waveform generator as shown below:

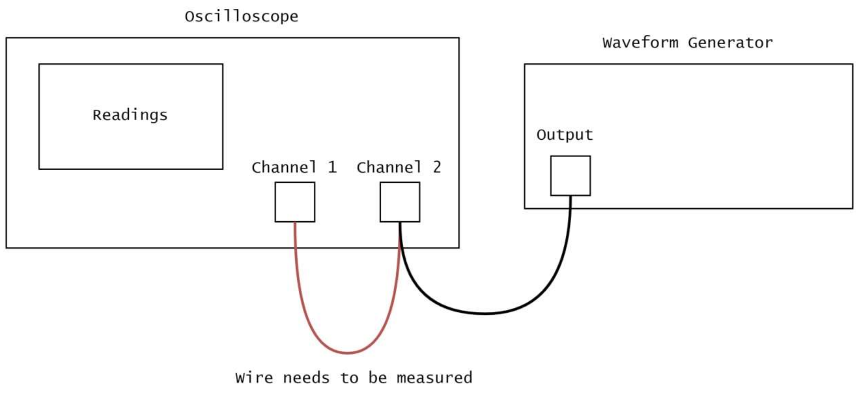


Fig. 9. The red wire is the wire that needs to be measured. The waveform generator first generates one square wave, and it reaches Channel 2 first, and then Channel 1.

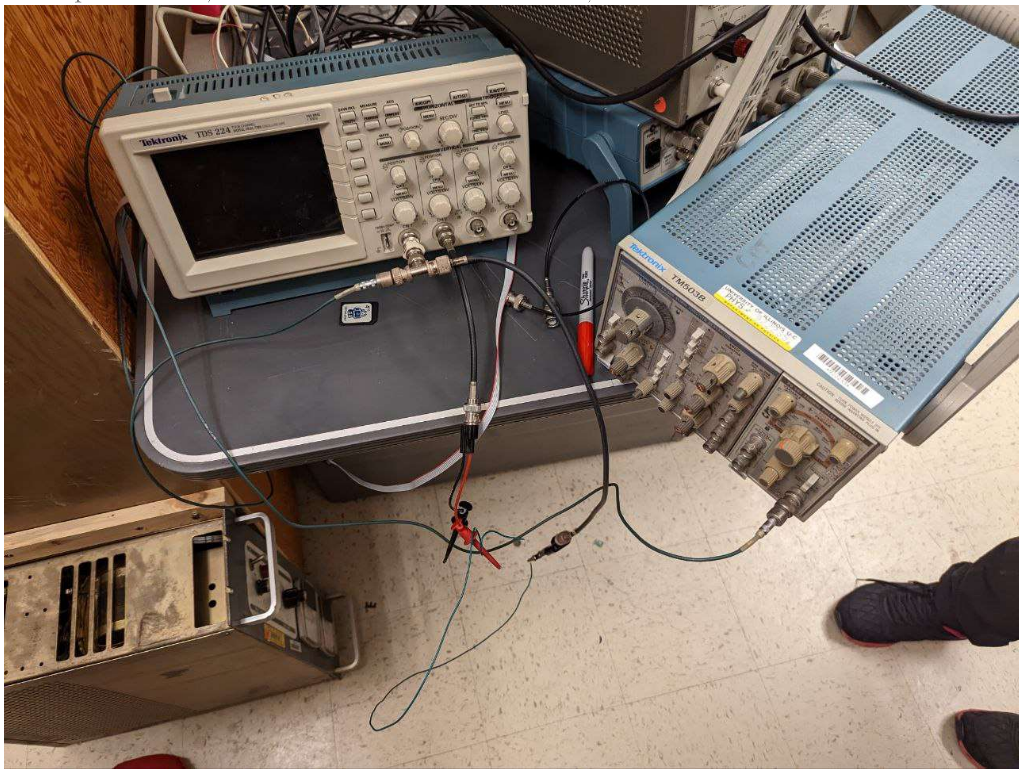


Fig. 10. This experiment setup of the Oscilloscope. The green thin wire is the wire that needs to be tested. The oscilloscope is on the left, the waveform generator is on the right.

The waveform generator will generate an oscillating square wave signal. The signal first reaches Channel 2, and then reaches Channel 1 via the wire that needs to be measured. The delay caused by the wire will reflect as the phase difference in the Oscilloscope readings, illustrated below:

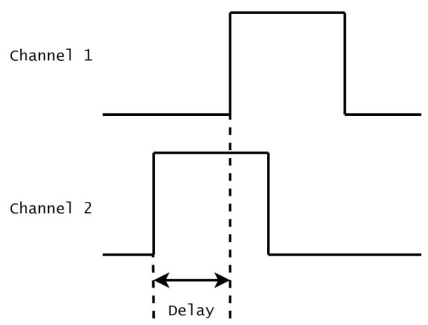


Fig. 11. This schematic and Fritzing graph for the accuracy-testing circuit

We could calculate the $v_{\text{Current Speed}}$ based on different length of the wire:

$$v_{\text{Current Speed}} = \text{avg}(\frac{x_{i_{\text{Wire Length}}}}{\Delta t_i(\text{Delay})})$$

Following the thought process, we set up a circuit with Arduino Mega 2560, SN74F74, and two Adafruit Ultimate GPS Breakout boards v3. The SN74F74 has two flipflop gates built in, and has slightly different pins and wiring, but it works similarly. We connect two flipflop gates two GPS modules, shown as below.

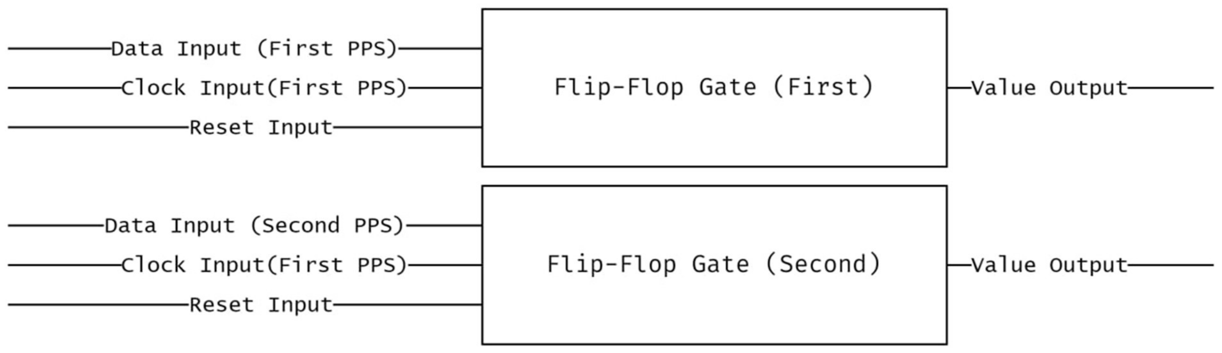


Fig. 12. The illustration of two flipflop gates that were used in the experiment.

The fritzing graph and schematic are shown below:

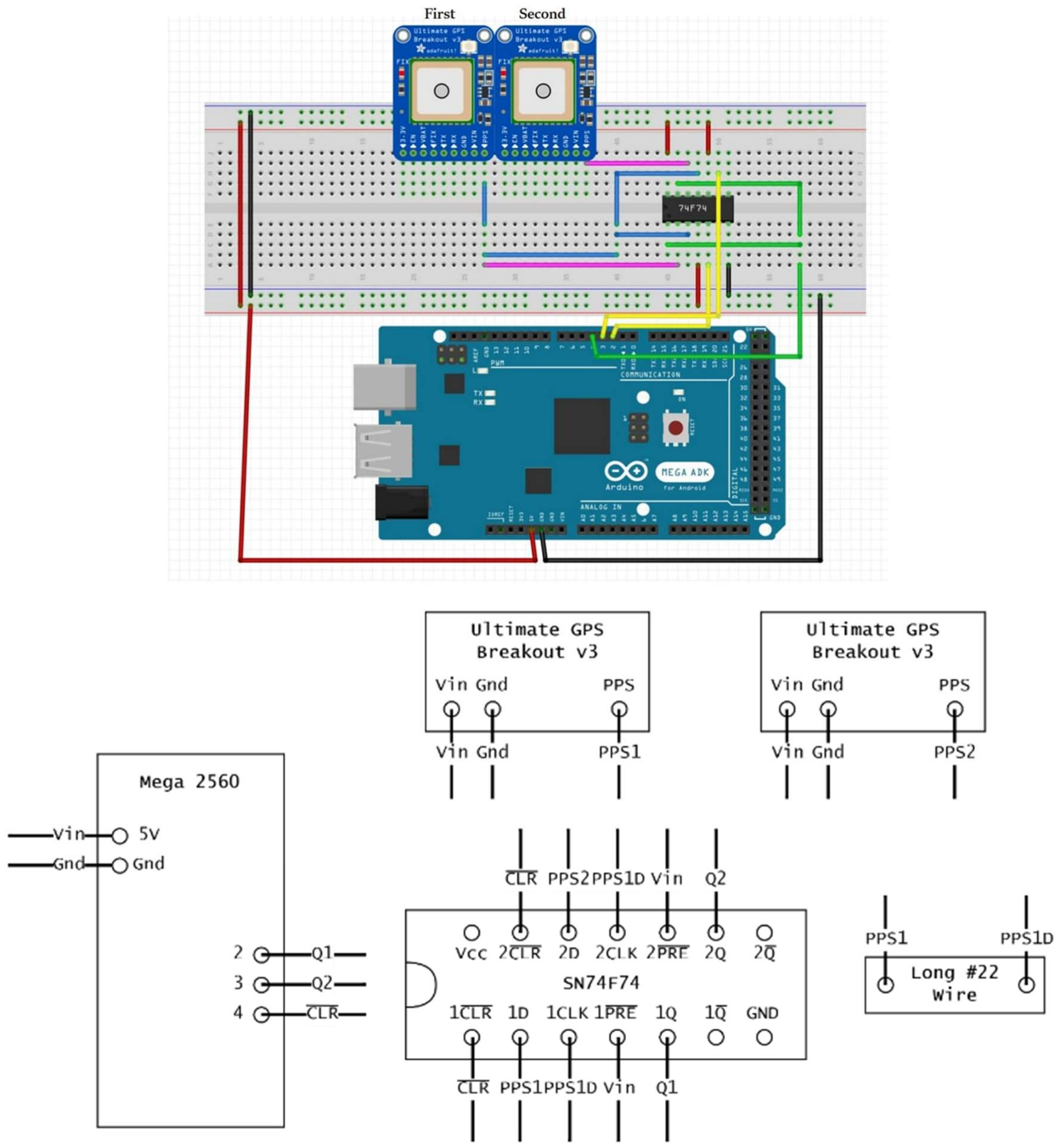


Fig. 13. The schematic and fritzing graph that is used to measure the PPS signal's time difference.

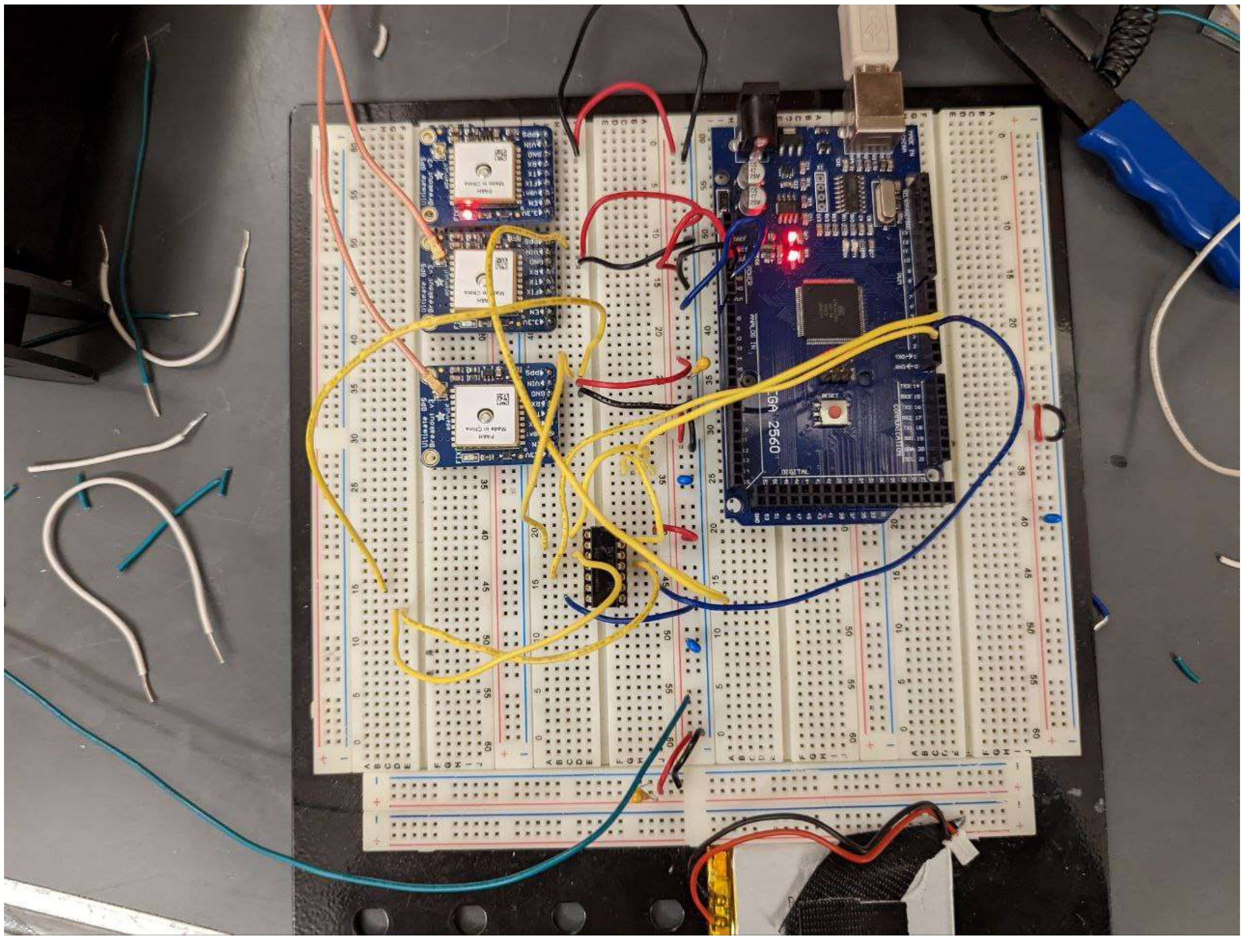
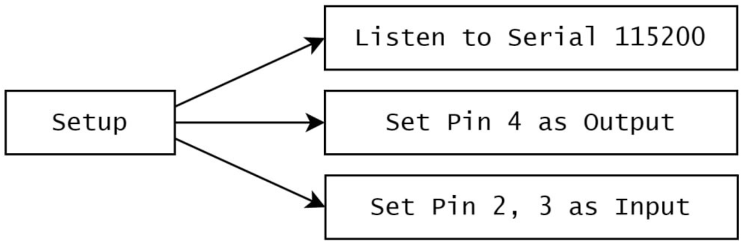


Fig. 14. The experiment setup. The small chip with yellow wires connected is the SN74F74, the GPU module is the top left (one is unused). They are connected to antennas to improve the connection with the satellite. The Arduino Mega 2560 is on the top right.

With these components set up, we write the data acquisition program into Mega 2560. The program reads the output pin (Q1, Q2) from the SN74F74, log it to serial, and reset the flipflop as required. Its execution could be seen below (code in Appendix III):



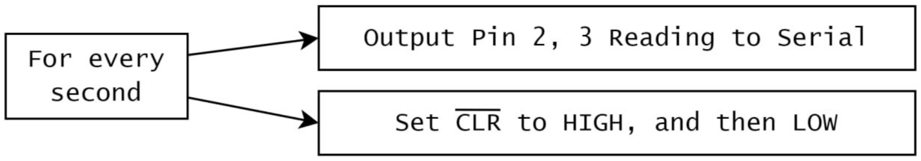

For every second
Output Pin 2, 3 Reading to Serial
Set $\overline{\text{CLR}}$ to HIGH, and then LOW

Fig. 15. The illustration of the program that is used to measure the time difference between PPS signals.

Result and Analysis

Internal Clock

For two Arduino Mega 2560, the data collected is shown below:

	$\langle \Delta t_{err} \rangle (us)$	$std(\Delta t)(us)$	p_{error}
Mega 2560 (Teal Color)	-1103.89	2.47	0.11%
Mega 2560 (Blue Color)	-7497.30	9.87	0.75%

Table 1. This table list (1) the average time difference between the internal clock, and the time indicated by PPS signal for one second (Negative means the internal clock is slower than PPS signal) (2) the corresponding standard deviation (3) the percent of time that is different from real time for 1 second.

We could see that the error varies greatly from different devices. Arduino Mega 2560 usually doesn't have an accurate internal clock: an error rate of 0.11% percent means that there will be 1s error per 10-20 minutes. However, Arduino Mega 2560 does have relative stability in their reading with a small standard deviation. This makes the previously mentioned simple calibration far more accurate.

After using the calibration method, the two Arduino Mega 2560 only differ in **4.0036us** on average. Considering their internal clocks' resolution is 4us, this level of synchronization is very accurate for Arduino Mega 2560.

The code used for data analysis can be found in Appendix IV. and V.

PPS Signal

The threshold wire length for both GPS modules is measured. Using oscilloscope, the current speed is found to be 82% of the light speed. So

	Threshold wire length (m)	Delayed Time (ns)
First GPS module	0.1286	0.522 (Minimum Setup Time)
Second GPS module	1.7018	6.918

Table 2. This table lists (1) the threshold wire length for the flipflop gate to detect input PPS signal. (2) the corresponding time delayed by the wire

The Minimum setup time is far less than the theoretical minimum setup time mentioned by the manufacturer. This might be due to manufacturing errors. The manufacturer might also give a very conservative minimum setup time to guarantee the quality.

The time difference in two GPS module is thus

$$\Delta t = 6.918 - 0.522 = 6.396\ (us)$$

This is aligned with the description from Adafruit that the PPS signal "have typical accuracy ranging 10ns".

Discussion

The experiment investigated the internal clock of Arduino Mega 2560, and found some interesting features as mentioned in the result section. However, the experiment could be done on a larger scale, with (1) more Arduino Mega 2560 board (2) with different Arduino board like Adafruit Feather M0 and Arduino Uno. Therefore, we could get more comprehensive data and understanding.

The experiment uses a simple approach to calibrate the Mega 2560 and synchronize them. It's better to use an approach that receives PPS signal and calibrates itself in real time. In scenario where the standard deviation of the error is large, this approach would perform far better than the naïve approach this report used.

When measuring the Minimum Setup Time, strictly speaking, the Minimum Setup Time should be different for two flipflop gates in the SN74F74. However, we assumed they were the same, and only measured the first one. Furthermore, when measuring the difference between PPS signals. The two GPS modules are actually put together. The GPS module calculates its time by location and other factors. Placing them together will give GPS module the same location and might reduce its error. Placing them far away might result in larger time error or different behaviors. Further research could be done with different setups.

Conclusion

This report investigated the accuracy of the internal clock of the Arduino Mega 2560 and the PPS signal from the Adafruit Ultimate GPS breakout v3. The result shows that Mega 2560's internal clock is inaccurate yet very stable. The accuracy of the PPS signal is aligned with the manufacturer documentation. The experiment could be improved by testing the internal clock of different Arduino boards and the PPS signal from GPS in different setups.

References

[1] Inc., G. T. (2022, 12 15). FGPMMOPA6H GPS Standalone Module Data Sheet. From https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf

[2] Instruments, T. (2022, 12 15). SN74F74 data sheet, product information and support. From Texas Instrument: https://www.ti.com/product/SN74F74

[3] micros() - Arduino Reference. (2022, 12 15). From Arduino: https://www.arduino.cc/reference/en/language/functions/time/micros/

[4] Pulse-per-second Signal. (2022, 12 15). From Wikipedia: https://en.wikipedia.org/wiki/Pulse-per-second_signal

Appendix

I. Arduino Mega 2560 Accuracy Test Code

```
#define PPS_PIN 9

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("TEST START");

  pinMode(PPS_PIN, INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  int pinValue = digitalRead(PPS_PIN);
  if (pinValue == HIGH) {
    Serial.println(micros());
  }
}
```

II. Arduino Mega 2560 "Send and Receive" code

Send Code

```
#define PPS_PIN 9
#define SEND_PIN 5

static unsigned long timer = 0;
static int index = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(SEND_PIN, OUTPUT);
  Serial.begin(115200);
  Serial.println("SEND START");
}

void loop() {
  unsigned long time = micros();
  // Serial.println(time - timer);
  if (time - timer > 1000000) {
    timer = time;
    digitalWrite(SEND_PIN, HIGH);
    digitalWrite(SEND_PIN, LOW);
    Serial.print('s');
    Serial.print(time);
    Serial.print(':');
    Serial.println(index);
    index++;
  }
}
```

Receive Code

```
#define PPS_PIN 9
#define RECEIVE_PIN A5

static unsigned long pulse_index = 0;
static uint8_t prevReceiveValue = LOW;

void setup() {
  // put your setup code here, to run once:
  pinMode(RECEIVE_PIN, INPUT);
  Serial.begin(115200);
  Serial.println("RECEIVE START");
}

void loop() {
  // put your main code here, to run repeatedly:
  // uint8_t ppsValue = digitalRead(PPS_PIN);
  uint8_t receiveValue = digitalRead(RECEIVE_PIN);

  if (prevReceiveValue == LOW && receiveValue == HIGH) {
    Serial.print('r');
    Serial.print(micros());
    Serial.print(':');
    Serial.println(pulse_index);
    pulse_index++;
  }

  prevReceiveValue = receiveValue;
}
```

III. PPS accuracy test code

```
#define CLR 4
#define Q1 2
#define Q2 3

void setup() {  // put your setup code here, to run once:
  pinMode(CLR, OUTPUT);
  Serial.begin(115200);

  digitalWrite(CLR, HIGH);
}

void loop() {
  // put your main code here, to run repeatedly:
    int valueOne = digitalRead(Q1);
    int valueTwo = digitalRead(Q2);
    Serial.print(valueOne);
    Serial.print(' ');
    Serial.println(valueTwo);

    // reset the state
    digitalWrite(CLR, LOW);
    delay(50);
    digitalWrite(CLR, HIGH);

    delay(950);
}
```

IV. Accuracy Test Analysis Code

```
const fs = require('fs');
const fileName = 'teal_second.log';

const raw = fs.readFileSync(fileName, 'utf8');
const data = raw.split('\n').map(Number);

const dataChunk = [[data[0]]]

for (const value of data) {
    // compare the last value of the last chunk with the current value
    const lastChunk = dataChunk[dataChunk.length - 1];
    const lastValue = lastChunk[lastChunk.length - 1];

    if (value - lastValue > 500000) {
        // if the difference is bigger than 500000, create a new chunk
        dataChunk.push([value]);
    } else {
        // otherwise add the value to the last chunk
        lastChunk.push(value);
    }
}
// pick the first one of each chunk
```

```
const pulseBeginMicros = dataChunk.map(chunk => chunk[0]);

// check the difference between them
const pulseDurations = pulseBeginMicros.map((value, index) => {
    if (index === 0) return 0;
    return value - pulseBeginMicros[index - 1];
}).map((val) => val - 1_000_000).filter((val) => Math.abs(val) <
100_000)

const diff = pulseDurations.slice(1);
const avg = diff.reduce((acc, val) => acc + val, 0) / diff.length;
const rms = Math.sqrt(diff.map((val) => (val - avg) ** 2).reduce((acc,
val) => acc + val, 0) / diff.length);

console.log("avg delay is", avg)
console.log("rms is", rms)
```

V. "Send & Receive" Calibration Analysis Code

```
const fs = require('fs');
const receiveFileName = "RECEIVE.log"

const raw = fs.readFileSync(receiveFileName, "utf8");
console.log(raw);

// the data has the form of "r[number]:[number]"
// extract both numbers
const rData = raw.split("\r\n").map((line) => {
    const [_, time, value] = line.match(/r(\d+):(\d+)/);
    return { time : Number(time), value: Number(value) };
});

// calc diff
const diff = rData.map((val, index) => {
    if (index === 0) return 0;
    return val.time - rData[index - 1].time;
}).slice(1).filter((val) => val <= 1_100_000)

// calc avg
const rAvg = diff.reduce((acc, val) => acc + val, 0) / diff.length;

const sendFileName = "SEND.log"
const sendRaw = fs.readFileSync(sendFileName, "utf8");

// the data has the form of "s[number]:[number]"
// extract both numbers
```

```
const sData = sendRaw.split("\r\n").map((line) => {
    const [_, time, value] = line.match(/s(\d+):(\d+)/);
    return { time : Number(time), value: Number(value) };
});

// calc diff
const sDiff = sData.map((val, index) => {
    if (index === 0) return 0;
    return val.time - sData[index - 1].time;
}).slice(1).filter((val) => val <= 1_100_000)

// calc avg
const sAvg = sDiff.reduce((acc, val) => acc + val, 0) / sDiff.length;

const R_CALI = -1103
const S_CALI = -7497

const R_DILATION =  1_000_000 / (1_000_000 + R_CALI)
const S_DILATION =  1_000_000 / (1_000_000 + S_CALI)

const rAvgCalibrated = rAvg * R_DILATION
const sAvgCalibrated = sAvg * S_DILATION

console.log("rAvg", rAvgCalibrated, "sAvg", sAvgCalibrated);
console.log("diff", rAvgCalibrated - sAvgCalibrated);
```

Raw data could be found within the resource folder.