2021 Quarknet Workshop

R. Armendariz and Steve Barton

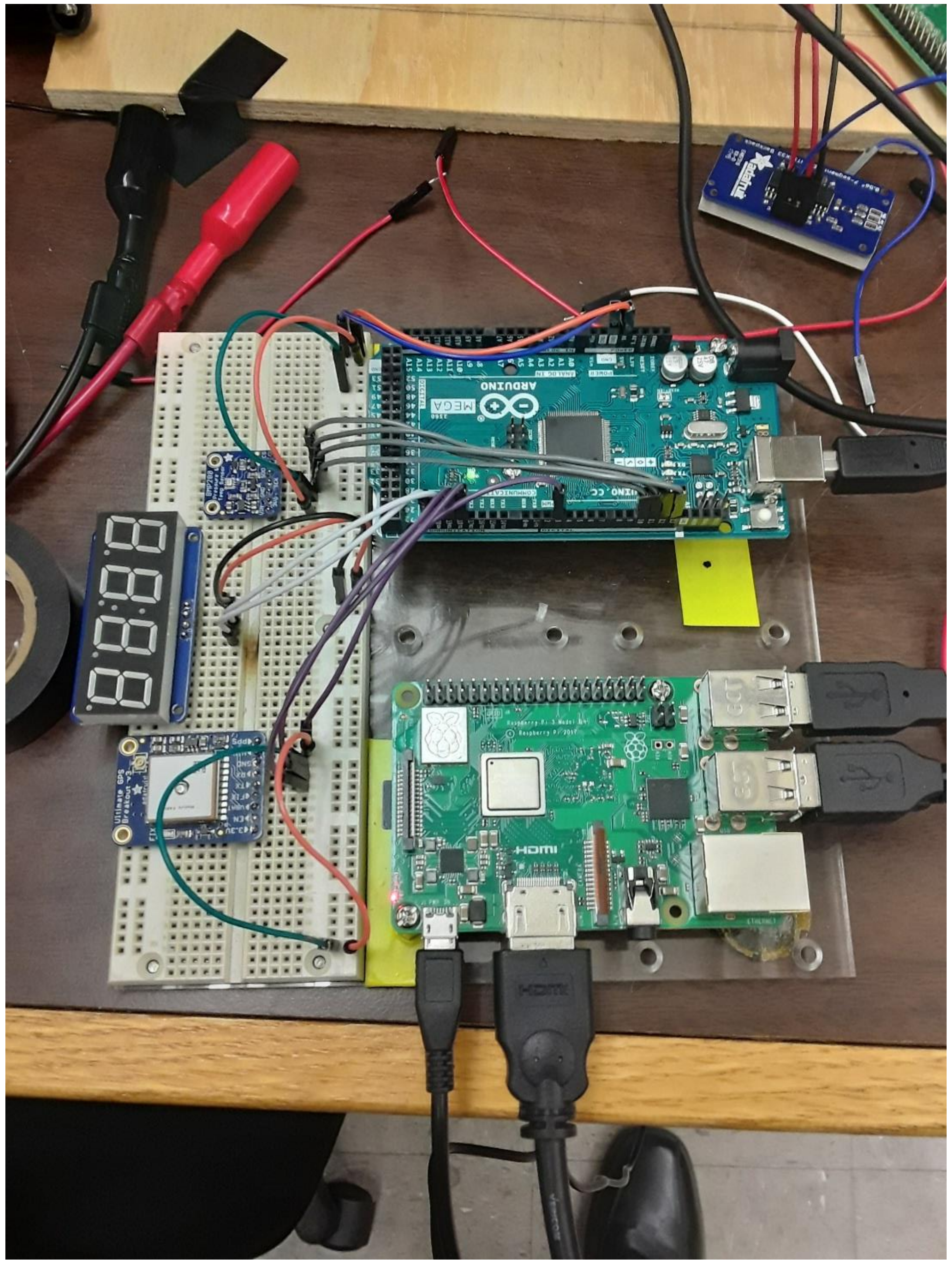
Arduino Mega programming tasks and GPS antenna/receiver measurements:

1) Create a square pulse signal using the function TimerOne(), and to read and plot the signal

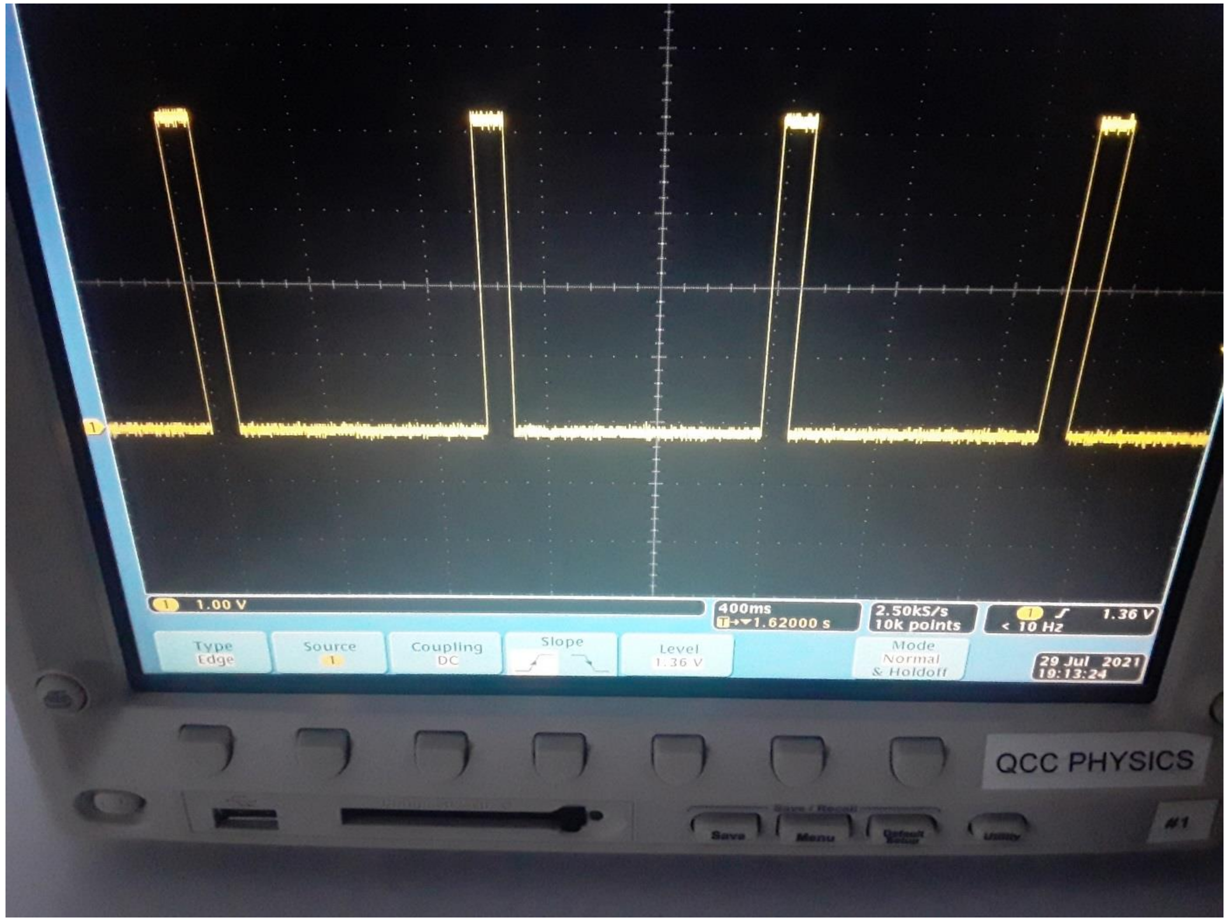
2) Record the GPS NMEA data

3) Measure the GPS antenna/receiver PPS square pulse and calculate the timing error

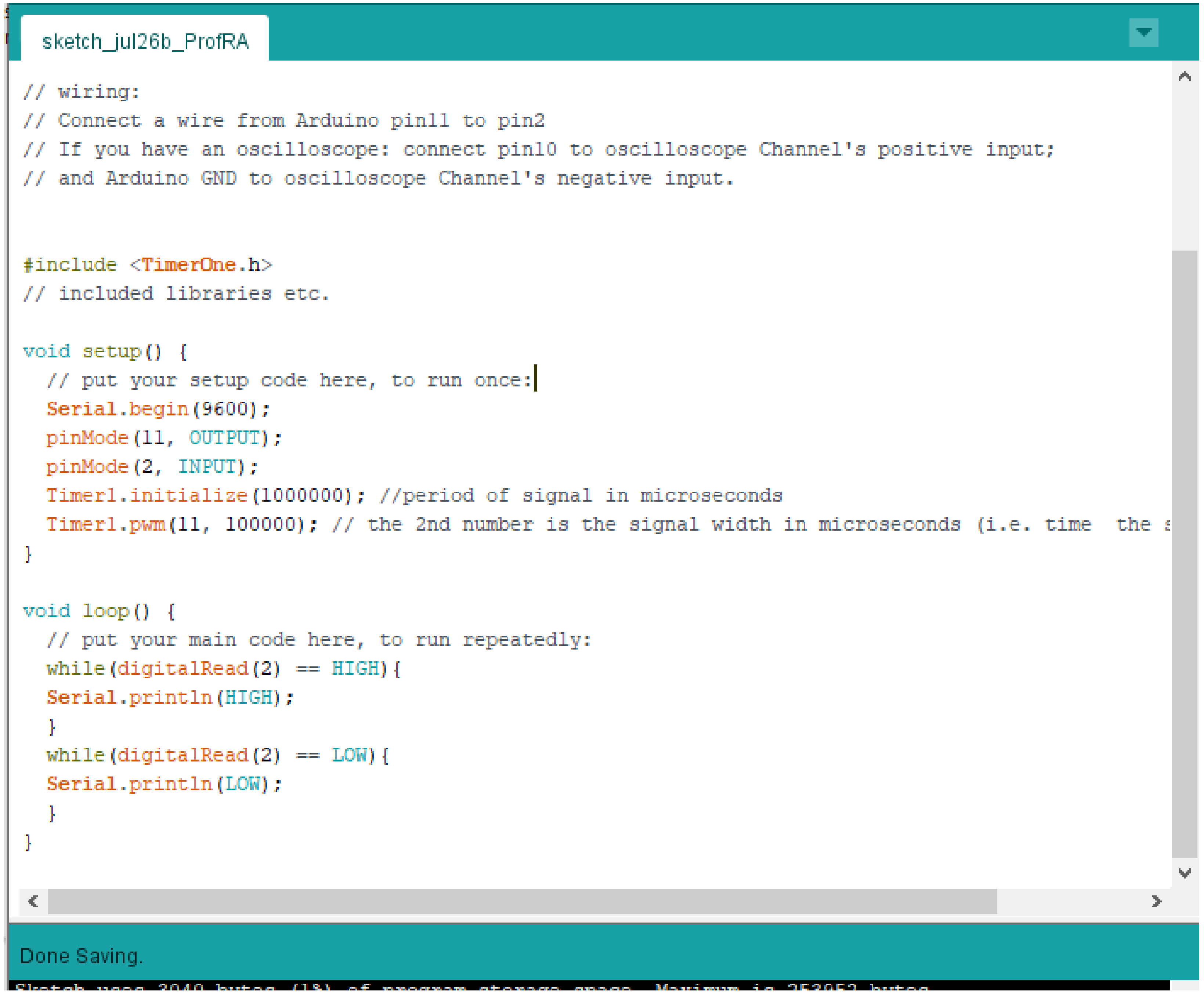
Arduino Mega shown connected to Raspberry Pi, GPS receiver, atmospheric pressure and temperature sensor, and digital counter



Square pulse with 1 sec period and 100 msec width generated with the Arduino Mega Timer1 function, shown on an oscilloscope



This code uses Timer1.initialize() and Timer1.pwm() to generate a square pulse from pin11 with a 1 second period, and 0.1 second width. It does not use the interrupt() or delay() functions

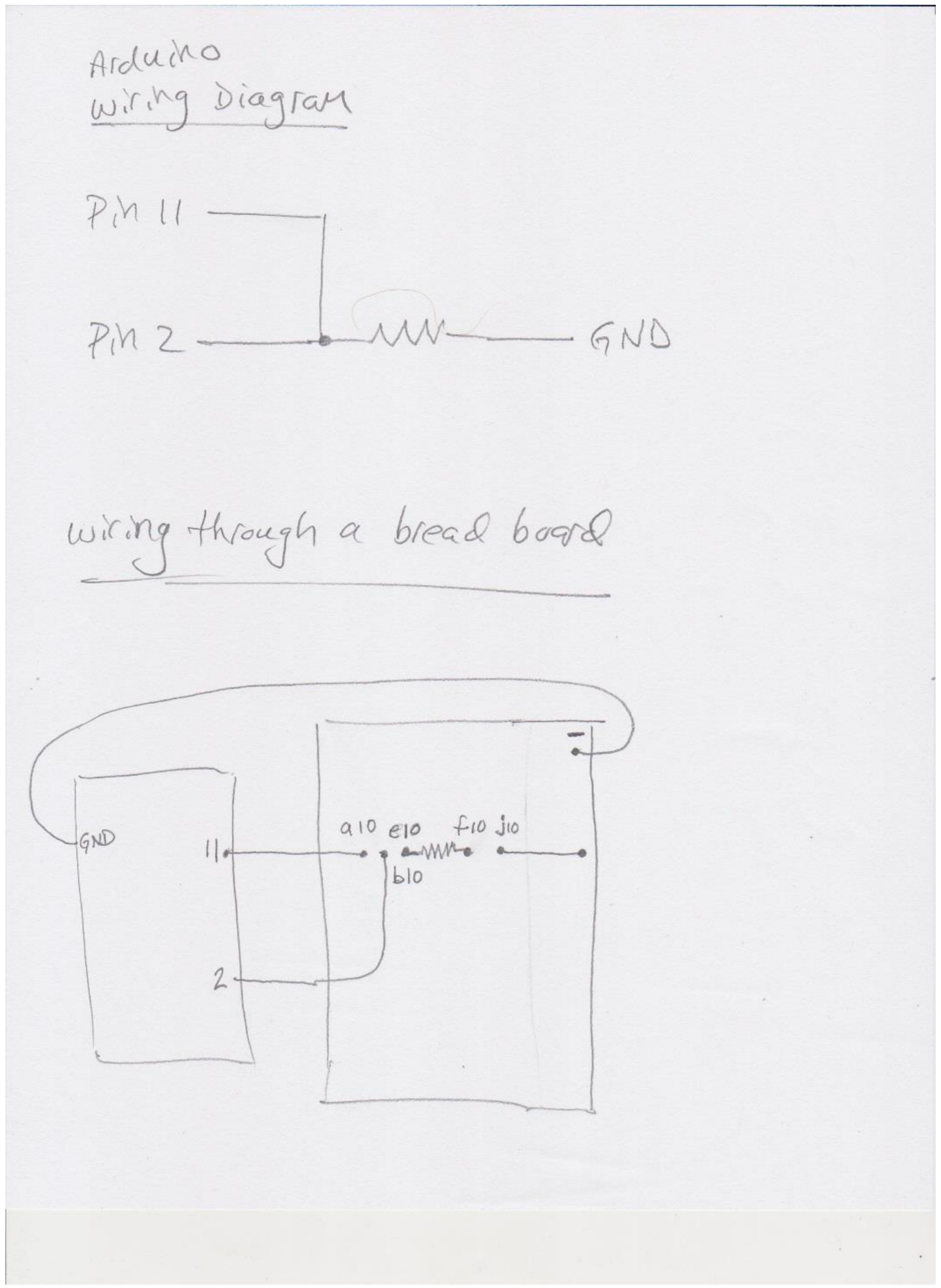

sketch_jul26b_ProfRA

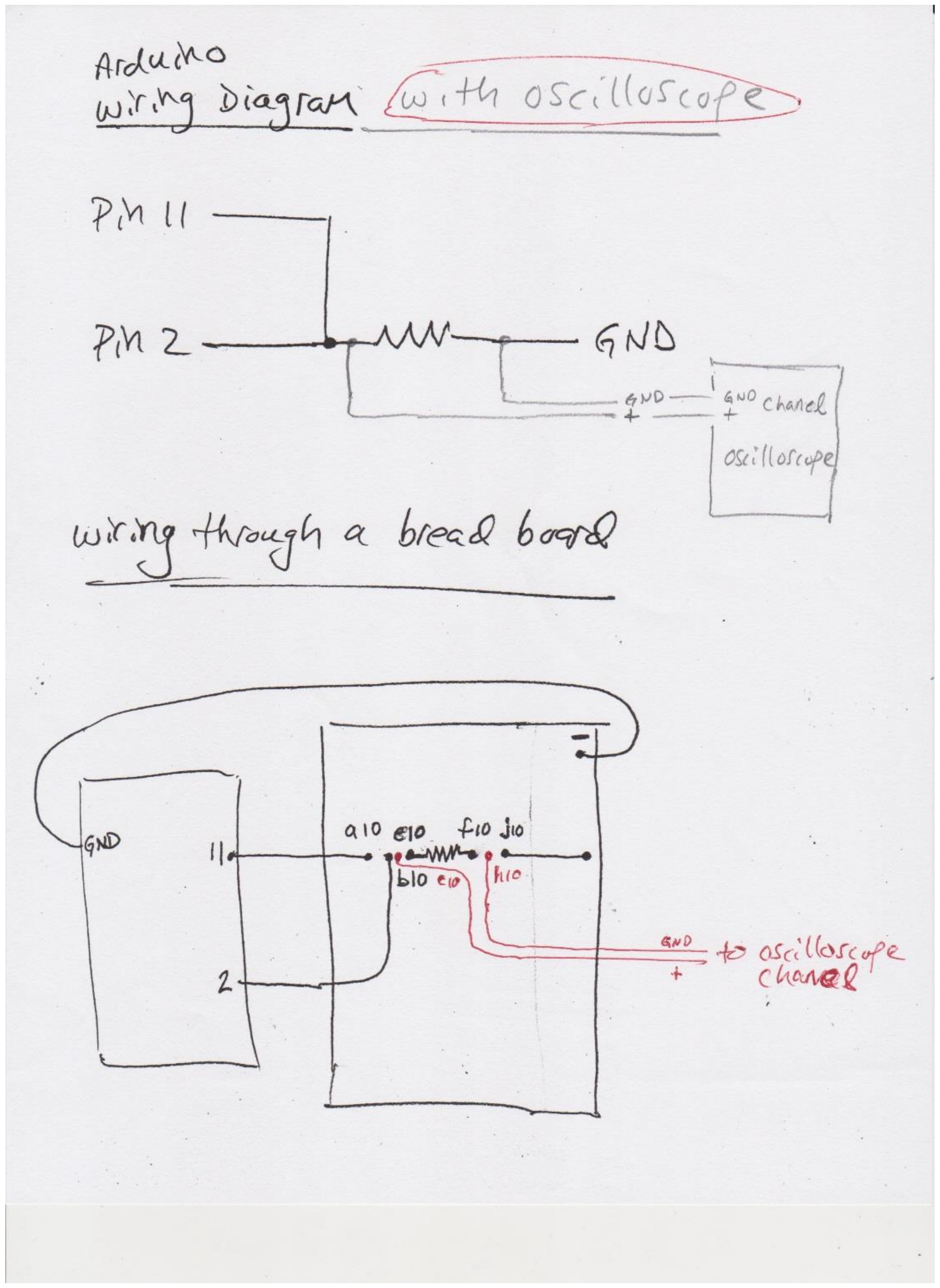
```
// wiring:
// Connect a wire from Arduino pin11 to pin2
// If you have an oscilloscope: connect pin10 to oscilloscope Channel's positive input;
// and Arduino GND to oscilloscope Channel's negative input.


#include <TimerOne.h>
// included libraries etc.

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(11, OUTPUT);
  pinMode(2, INPUT);
  Timer1.initialize(1000000); //period of signal in microseconds
  Timer1.pwm(11, 100000); // the 2nd number is the signal width in microseconds (i.e. time  the s
}

void loop() {
  // put your main code here, to run repeatedly:
  while(digitalRead(2) == HIGH){
  Serial.println(HIGH);
  }
  while(digitalRead(2) == LOW){
  Serial.println(LOW);
  }
}
```

Done Saving.

```
Sketch uses 3040 bytes (1%) of program storage space. Maximum is 253952 bytes.
```

A wire connects pin11 to pin2; a hold-down resistor connects them to ground to ensure the voltage on pin2 does not float.





```
14:19:40.602 -> 0
14:19:40.602 -> 0
14:19:40.602 -> 0
14:19:40.649 -> 0
14:19:40.649 -> 0
14:19:40.649 -> 0
14:19:40.649 -> 0
signal goes high at 14:19:40.649
14:19:40.649 -> 1
14:19:40.649 -> 1
14:19:40.649 -> 1
14:19:40.649 -> 1
14:19:40.649 -> 1
14:19:40.649 -> 1
14:19:40.649 -> 1
14:19:40.649 -> 1
14:19:40.649 -> 1
14:19:40.649 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
```

```
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.696 -> 1
14:19:40.743 -> 1
14:19:40.743 -> 1
14:19:40.743 -> 1
14:19:40.743 -> 1
14:19:40.743 -> 1
14:19:40.743 -> 1
signal goes low at 14:19:40.743
14:19:40.743 -> 0
14:19:40.743 -> 0
14:19:40.743 -> 0
14:19:40.743 -> 0
Data continued in 47ms increments
```

```
14:19:41.587 -> 0
14:19:41.587 -> 0
14:19:41.587 -> 0
14:19:41.587 -> 0
14:19:41.634 -> 0
14:19:41.634 -> 0
14:19:41.634 -> 0
14:19:41.634 -> 0
14:19:41.634 -> 0
14:19:41.634 -> 0
14:19:41.634 -> 0
14:19:41.634 -> 0
signal goes high again at 14:19:41.634
14:19:41.634 -> 1
14:19:41.634 -> 1
14:19:41.634 -> 1
14:19:41.634 -> 1
14:19:41.634 -> 1
14:19:41.634 -> 1
14:19:41.681 -> 1
14:19:41.681 -> 1
14:19:41.681 -> 1
14:19:41.681 -> 1
14:19:41.681 -> 1
14:19:41.681 -> 1
```

Description of Serial monitor data

The data shows the recorded pulse shape is off by 6 ms in width and 15 ms in period; more investigation is needed to find the source of this problem

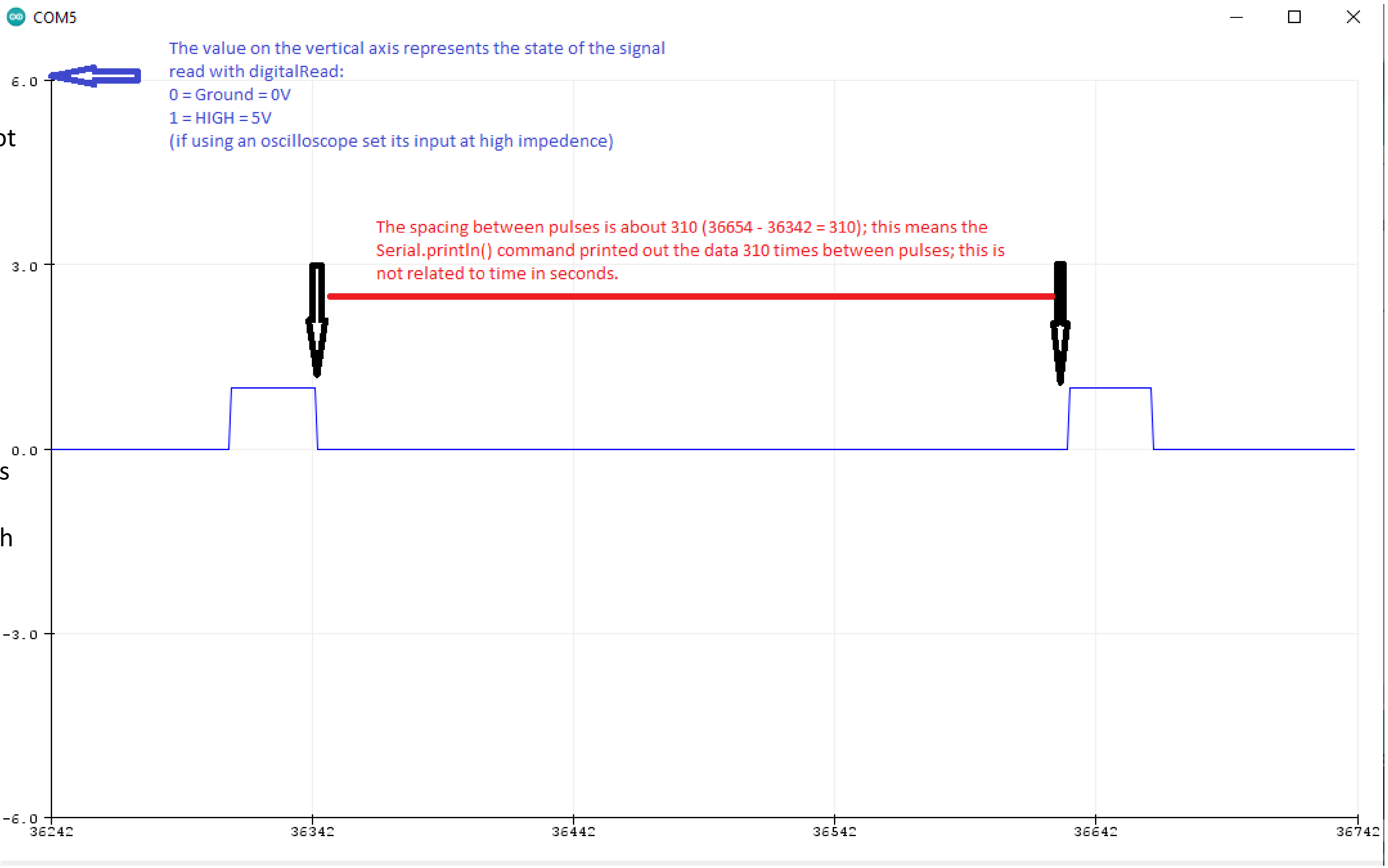
Signal width = high - low = 40.743 - 40.649 = 94 ms

Signal period =
high2 - high1 = 41.634 - 40.649 = 985 ms

At each time stamp the code prints out several lines of data. The time stamps typically (but not always) increase in 47 millisecond increments, i.e. 14:19:40.602 jumps to 14:19:40.649.

Serial plotter plot

The serial plotter plot is correct; its horizontal axis represents the number of consecutive print lines from Serial.println(). The signal's time period cannot be interpreted from this plot, however it is clear the signal width is about 10% of the signal period.



The value on the vertical axis represents the state of the signal read with digitalRead:
0 = Ground = 0V
1 = HIGH = 5V
(if using an oscilloscope set its input at high impedence)

The spacing between pulses is about 310 (36654 - 36342 = 310); this means the Serial.println() command printed out the data 310 times between pulses; this is not related to time in seconds.

EXCEL HELP to make the square pulse plot in Excel (the plot is shown on next slide)

Copy and paste several seconds worth of serial monitor data into Excel, don't copy the first few seconds because sometimes the Arduino takes a few seconds to stabilize;

The data looks like this:
11:18:04.411 -> 0

11:18:04.411 is the time stamp (04.411 means 4 seconds and 411 ms), and 0 is the signal amplitude

Separate the time and amplitude into two separate columns, keep only the seconds and millisconds, and the amplitude; I did it this way:

https://support.microsoft.com/en-us/office/split-a-cell-f1804d0c-e180-4ed0-a2ae-973a0b7c6a23

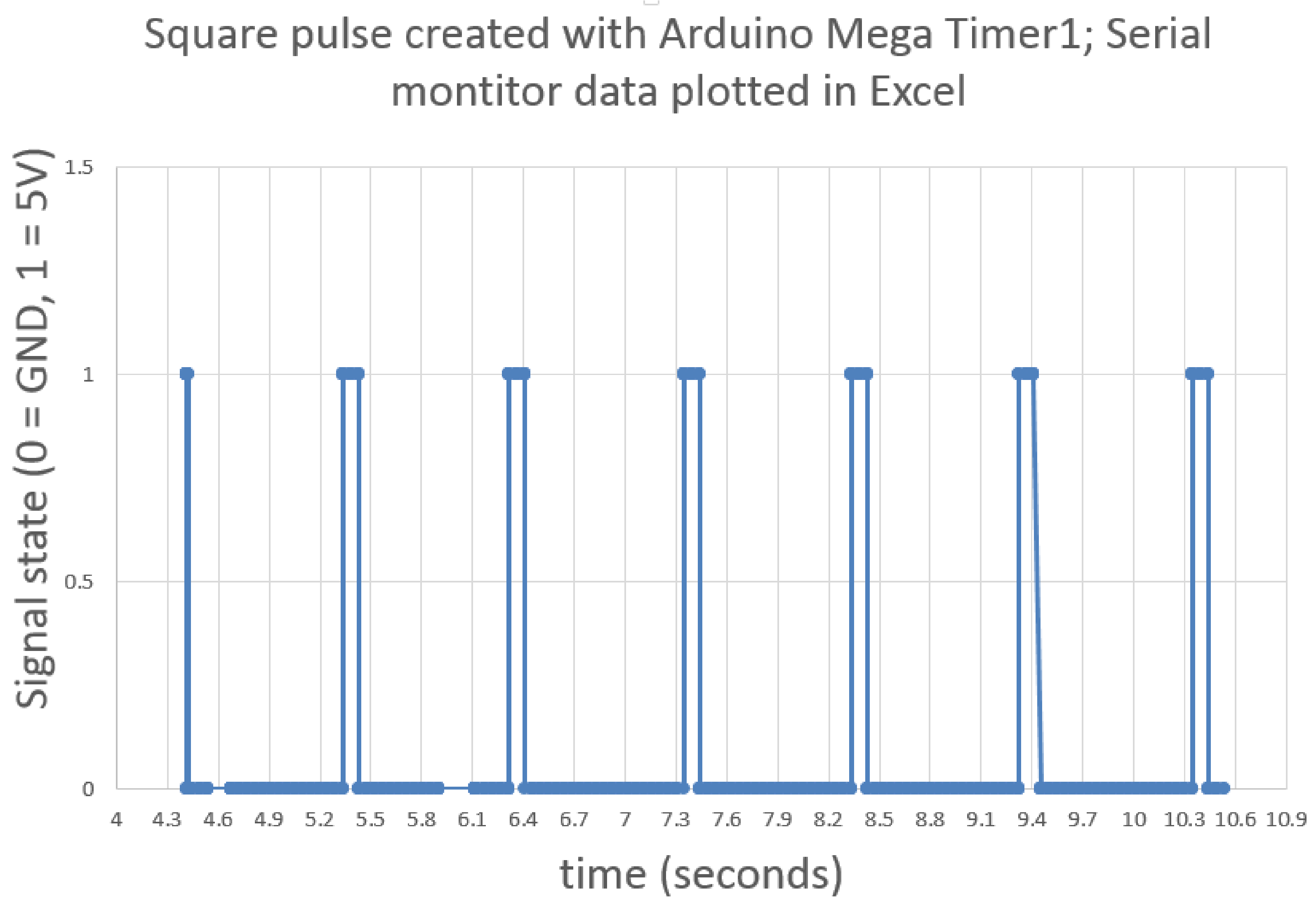
separate this: 11:18:04.411 -> 0
like this: 11:18: 04.411-> 0

then separate this: 04.411-> 0
like this 04.411 -> 0

keep only the columns: 04.411 0
make the plot

If Excel rounds the numbers in the cells you have to change the settings in Home -> Number -> Category(Numbers) -> decimal places (increase it to 3, on positive numbers);

If Excel rounds the numbers on the plot axis you can prevent that on the plot's axis settings



The serial data plotted shows the signal read by the Arduino is correct; here the horizontal axis shows the pulse time; the data is from Serial.println()

To record the GPS receiver NMEA data in the Arduino serial monitor:

wire together:

Arduino 5V to Vin

Arduino GND to GPS GND

Arduino RX0 to GPS RX

Arduino TX0 to GPS TX

Run this empty code:

```
void setup(){
}
void loop(){
}
```

To record the GPS receiver PPS square pulse in the Arduino serial monitor and serial plotter:

wire together:

Arduino 5V to GPS Vin

Arduino GND to GPS GND

Arduino PWM 2 to GPS PPS

Run this code:

```
void setup(){
Serial.begin(9600);
pinMode (2, INPUT);
}
void loop(){
while(digitalRead(2) == HIGH){
Serial.println(HIGH);
}
while(digitalRead(2) == LOW){
Serial.println(LOW);
}
}
```

To calculate and plot the error on the PPS signal do the following (plot is shown on next slide):

Collect serial monitor data for several consecutive PPS pulses; each pulse has a pulse start time when it goes high, and a pulse stop time when it goes low; in Excel calculate the time difference between each pair of consecutive pulses and make a table of at least 25 data points like this:

Data point 1 = 1 - [(pulse 2 start time) - (pulse 1 start time)]

Data point 2 = 1 - [(pulse 3 start time) - (pulse 2 start time)]

Data point 3 = 1 - [(pulse 4 start time) - (pulse 3 start time)]

Data point 4 = 1 - [(pulse 5 start time) - (pulse 4 start time)]

Etc. etc. …

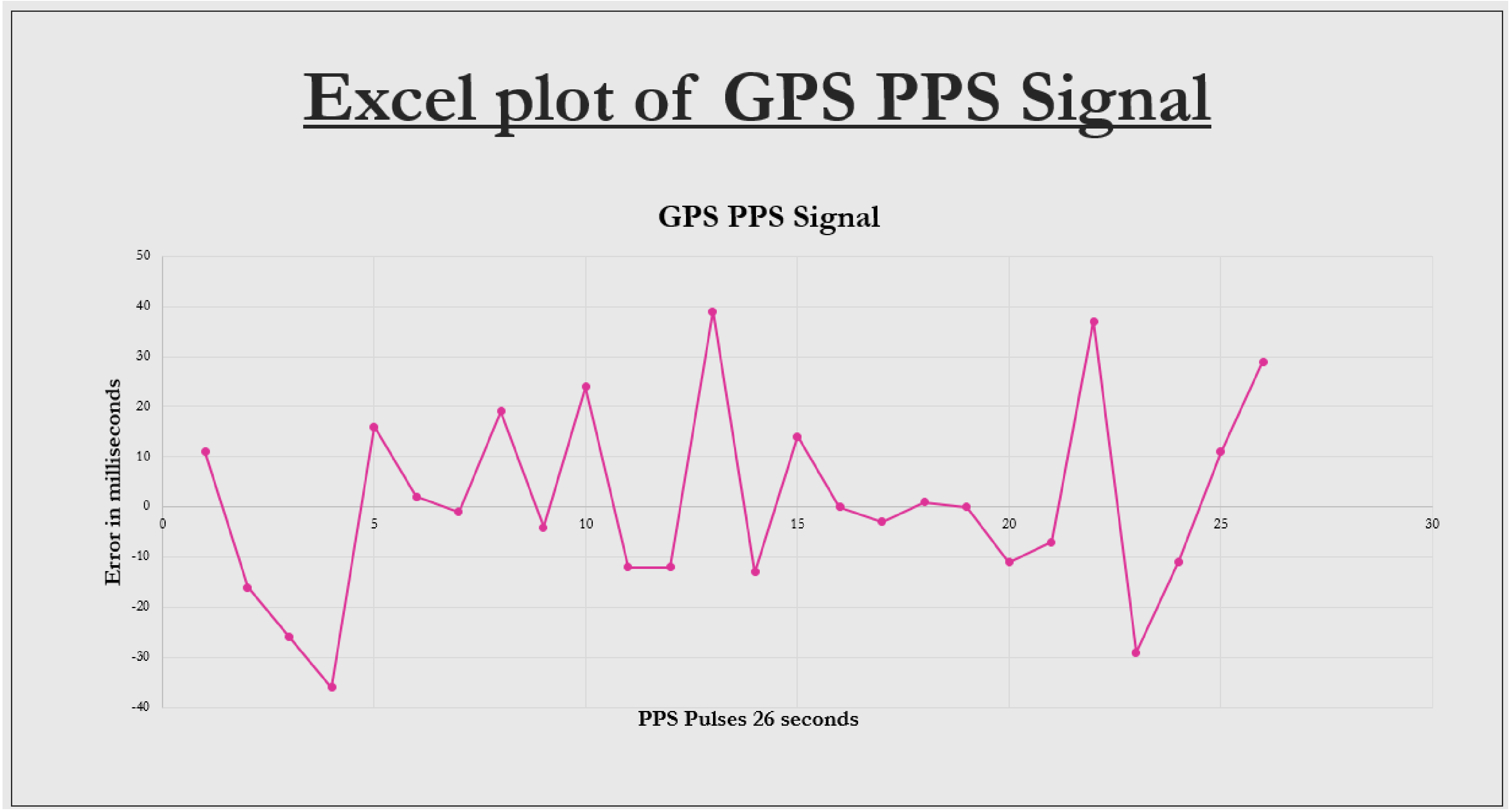
Data point 25 = 1 - [(pulse 26 start time) - (pulse 25 start time)]

But only use the seconds and milliseconds; for example for the two pulses 14:19:40.649 and 14:19:41.634 use:

Data point 1 - [(41.634) - (40.649)] = 0.015 seconds

In Excel make a plot of this data; the horizontal axis goes from 1 through 25, label it "PPS pulses over 26 seconds." Label the vertical axis "Timing error between consecutive PPS pulses in milliseconds." Draw the plot's vertical axis in 5 or 10 millisecond increments to clearly see the data points fluctuate up and down about the zero line.

Plot shows the timing error on the GPS PPS signal period:
error = 1000 msec – (difference between the start times of consecutive pulses)



You can also use a pulse start time and end time to calculate the percent error in pulse width from the expected value of 100 milliseconds; and use the pulse start times for two consecutive pulses to calculate the percent error in the period from the expected value of 1 second.