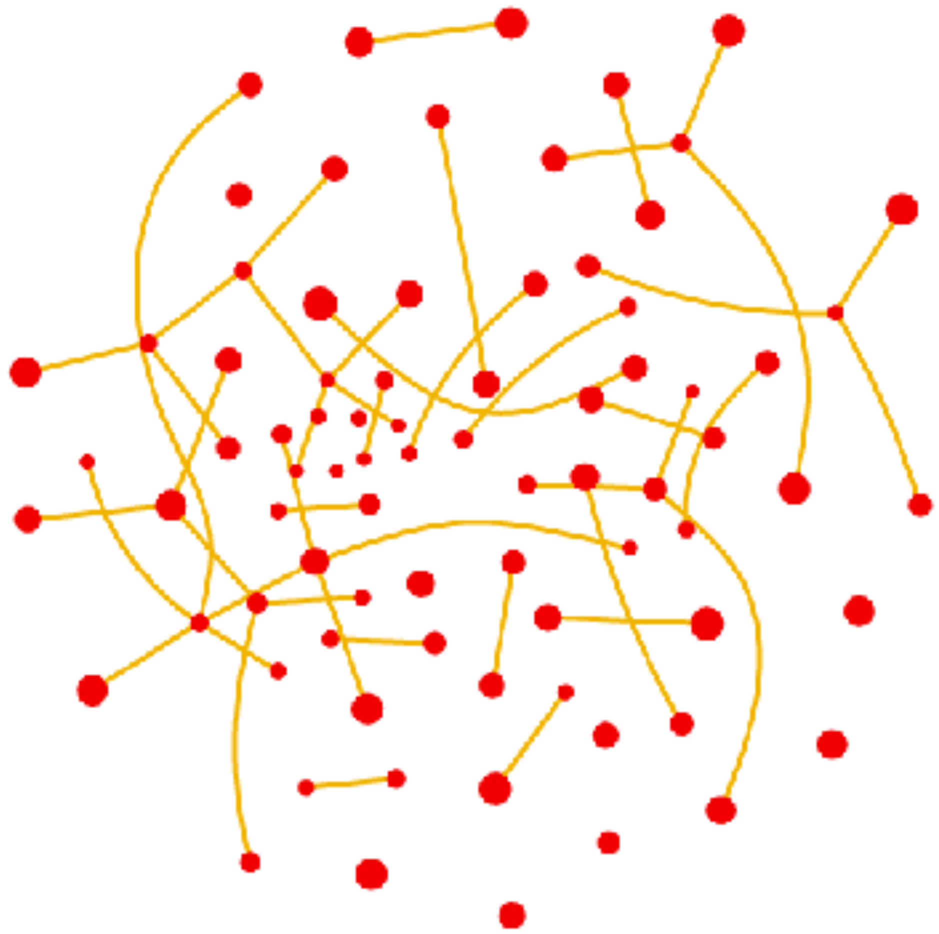


Working with Sensors
Part II

Working with sensor data

Garbage in

Garbage out

What we’ve covered

Garbage in / garbage out - the need to know your sensor

Calibration

Collecting sample data

What we'll cover

Know your sensor - Reading a data sheet

Smoothing and Noise

Algorithms for Data Sensemaking

Limits of a Photon

Know your Sensor

Or how to read a data sheet and why you might want to

What is a data sheet

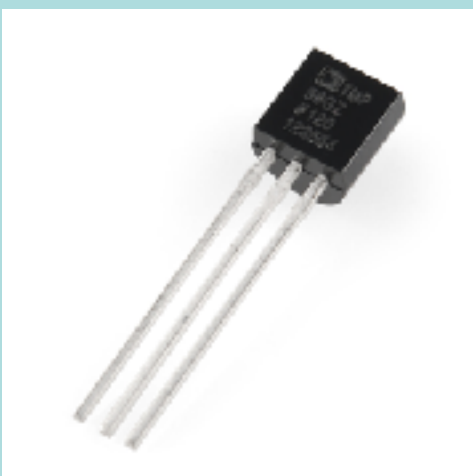
It's a manual for a component

It tells you everything (often in too much detail) you need to know

- device performance
- operation, requirements and characteristics
- tolerances and how to harm it
- suggested uses and hints

Know your Sensor

Temperature Sensor - TMP36



ANALOG DEVICES

Low Voltage Temperature Sensors

TMP35/TMP36/TMP37

FEATURES

Low voltage operation (2.7 V to 5.5 V)
Calibrated directly in °C
10 mV/°C scale factor (20 mV/°C on TMP37)
±2°C accuracy over temperature (typ)
±0.5°C linearity (typ)
Stable with large capacitive loads
Specified −40°C to +125°C, operation to +150°C
Less than 50 µA quiescent current
Shutdown current 0.5 µA max
Low self-heating
Qualified for automotive applications

APPLICATIONS

Environmental control systems
Thermal protection
Industrial process control
Fire alarms
Power system monitors
CPU thermal management

GENERAL DESCRIPTION

The TMP35/TMP36/TMP37 are low voltage, precision centigrade temperature sensors. They provide a voltage output that is linearly proportional to the Celsius (centigrade) temperature. The TMP35/ TMP36/TMP37 do not require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C over the −40°C to +125°C temperature range.

The low output impedance of the TMP35/TMP36/TMP37 and its linear output and precise calibration simplify interfacing to temperature control circuitry and ADCs. All three devices are intended for single-supply operation from 2.7 V to 5.5 V maximum. The supply current runs well below 50 µA, providing very low self-heating—less than 0.1°C in still air. In addition, a shutdown function is provided to cut the supply current to less than 0.5 µA.

The TMP35 is functionally compatible with the LM35/LM45 and provides a 250 mV output at 25°C. The TMP35 reads temperatures from 10°C to 125°C. The TMP36 is specified from −40°C to +125°C, provides a 750 mV output at 25°C, and operates to 125°C from a single 2.7 V supply. The TMP36 is functionally compatible with the LM50. Both the TMP35 and TMP36 have an output scale factor of 10 mV/°C.

FUNCTIONAL BLOCK DIAGRAM

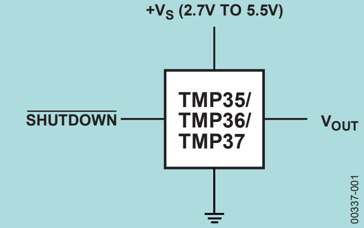


Figure 1.

PIN CONFIGURATIONS

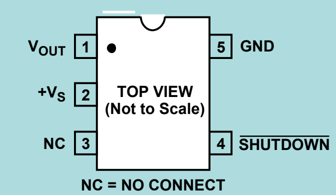


Figure 2. RJ-5 (SOT-23)

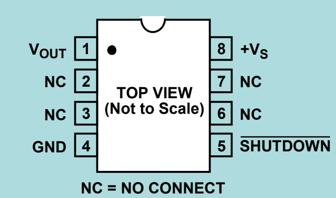


Figure 3. R-8 (SOIC_N)

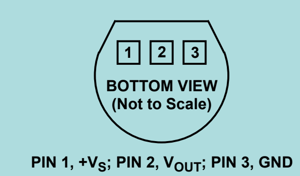


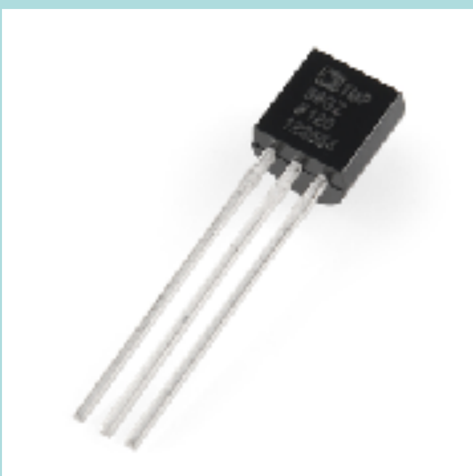
Figure 4. T-3 (TO-92)

The TMP37 is intended for applications over the range of 5°C to 100°C and provides an output scale factor of 20 mV/°C. The TMP37 provides a 500 mV output at 25°C. Operation extends to 150°C with reduced accuracy for all devices when operating from a 5 V supply.

The TMP35/TMP36/TMP37 are available in low cost 3-lead TO-92, 8-lead SOIC_N, and 5-lead SOT-23 surface-mount packages.

Know your Sensor

Temperature Sensor - TMP36



ANALOG DEVICES

Low Voltage Temperature Sensors

TMP35/TMP36/TMP37

FEATURES

Low voltage operation (2.7 V to 5.5 V)
Calibrated directly in °C
10 mV/°C scale factor (20 mV/°C on TMP37)
±2°C accuracy over temperature (typ)
±0.5°C linearity (typ)
Stable with large capacitive loads
Specified −40°C to +125°C, operation to +150°C
Less than 50 µA quiescent current
Shutdown current 0.5 µA max
Low self-heating
Qualified for automotive applications

APPLICATIONS

Environmental control systems
Thermal protection
Industrial process control
Fire alarms
Power system monitors
CPU thermal management

GENERAL DESCRIPTION

The TMP35/TMP36/TMP37 are low voltage, precision centigrade temperature sensors. They provide a voltage output that is linearly proportional to the Celsius (centigrade) temperature. The TMP35/ TMP36/TMP37 do not require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C over the −40°C to +125°C temperature range.

The low output impedance of the TMP35/TMP36/TMP37 and its linear output and precise calibration simplify interfacing to temperature control circuitry and ADCs. All three devices are intended for single-supply operation from 2.7 V to 5.5 V maximum. The supply current runs well below 50 µA, providing very low self-heating—less than 0.1°C in still air. In addition, a shutdown function is provided to cut the supply current to less than 0.5 µA.

The TMP35 is functionally compatible with the LM35/LM45 and provides a 250 mV output at 25°C. The TMP35 reads temperatures from 10°C to 125°C. The TMP36 is specified from −40°C to +125°C, provides a 750 mV output at 25°C, and operates to 125°C from a single 2.7 V supply. The TMP36 is functionally compatible with the LM50. Both the TMP35 and TMP36 have an output scale factor of 10 mV/°C.

FUNCTIONAL BLOCK DIAGRAM

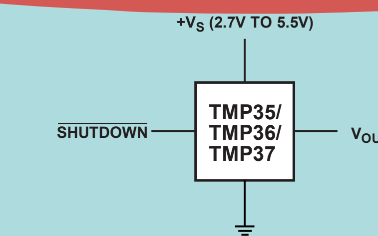


Figure 1.

PIN CONFIGURATIONS

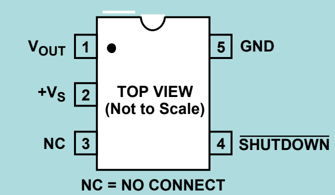


Figure 2. RJ-5 (SOT-23)

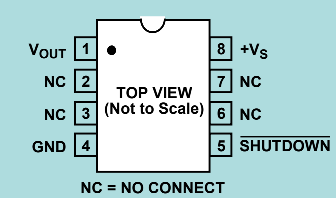


Figure 3. R-8 (SOIC_N)

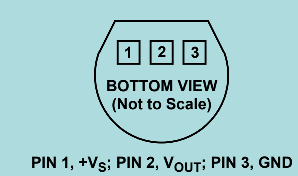


Figure 4. T-3 (TO-92)

The TMP37 is intended for applications over the range of 5°C to 100°C and provides an output scale factor of 20 mV/°C. The TMP37 provides a 500 mV output at 25°C. Operation extends to 150°C with reduced accuracy for all devices when operating from a 5 V supply.

The TMP35/TMP36/TMP37 are available in low cost 3-lead TO-92, 8-lead SOIC_N, and 5-lead SOT-23 surface-mount packages.

Know your Sensor

Temperature Sensor - TMP36

PIN CONFIGURATIONS

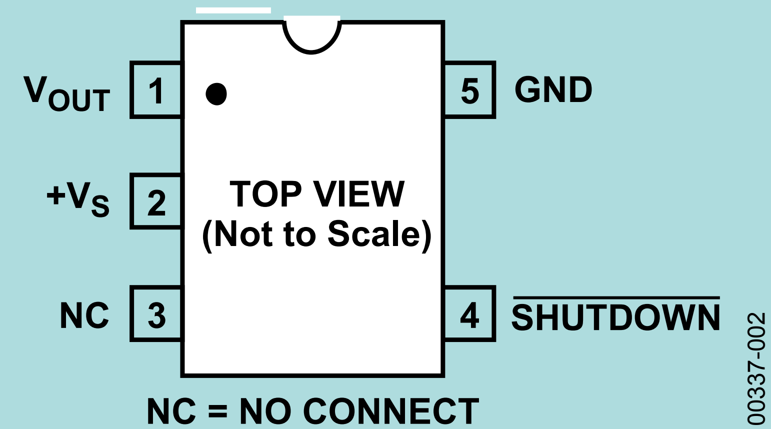


Figure 2. RJ-5 (SOT-23)

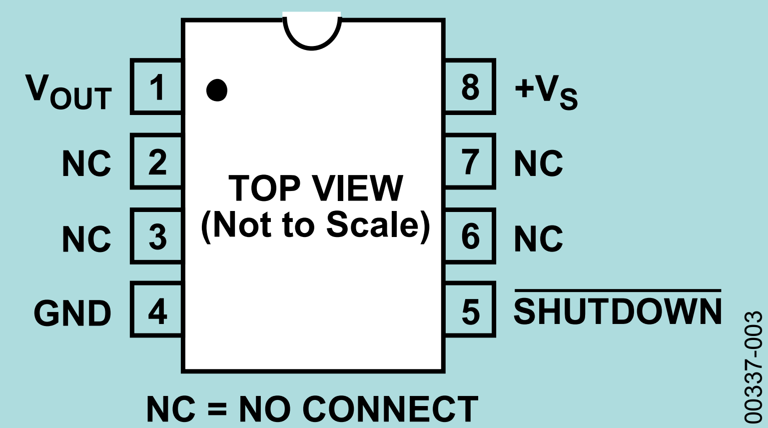


Figure 3. R-8 (SOIC_N)

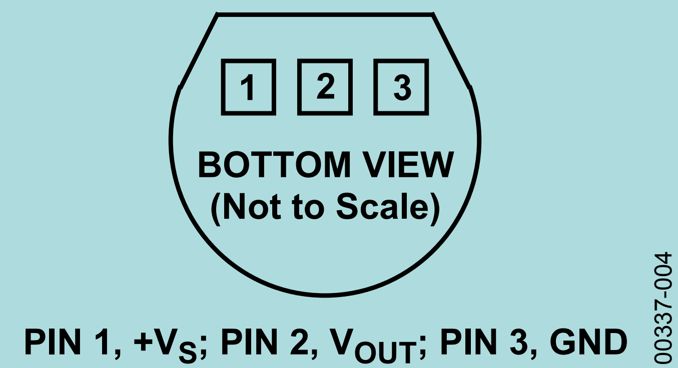


Figure 4. T-3 (TO-92)

Know your Sensor

Temperature Sensor - TMP36

FEATURES

Low voltage operation (2.7 V to 5.5 V)
Calibrated directly in °C
10 mV/°C scale factor (20 mV/°C on TMP37)
±2°C accuracy over temperature (typ)
±0.5°C linearity (typ)
Stable with large capacitive loads
Specified −40°C to +125°C, operation to +150°C
Less than 50 μA quiescent current
Shutdown current 0.5 μA max
Low self-heating
Qualified for automotive applications

APPLICATIONS

Environmental control systems
Thermal protection
Industrial process control
Fire alarms
Power system monitors
CPU thermal management

Know your Sensor

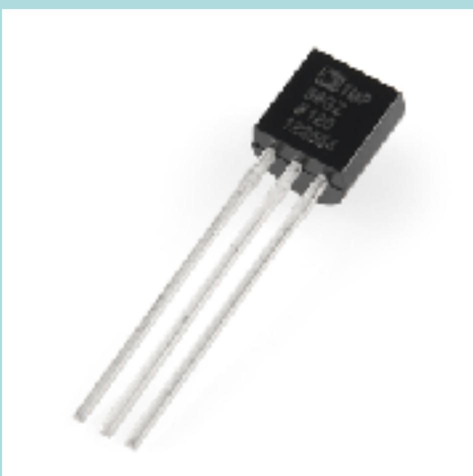
Temperature Sensor - TMP36

The TMP35/TMP36/TMP37 are low voltage, precision centigrade temperature sensors. They provide a voltage output that is linearly proportional to the Celsius (centigrade) temperature. The TMP35/ TMP36/TMP37 do not require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C over the −40°C to +125°C temperature range.

and provides a 250 mV output at 25°C. The TMP35 reads temperatures from 10°C to 125°C. The TMP36 is specified from −40°C to +125°C, provides a 750 mV output at 25°C, and operates to 125°C from a single 2.7 V supply. The TMP36 is functionally compatible with the LM50. Both the TMP35 and TMP36 have an output scale factor of 10 mV/°C.

Know your Sensor

Temperature Sensor - TMP36



ANALOG DEVICES

Low Voltage Temperature Sensors

TMP35/TMP36/TMP37

FEATURES

Low voltage operation (2.7 V to 5.5 V)
Calibrated directly in °C
10 mV/°C scale factor (20 mV/°C on TMP37)
±2°C accuracy over temperature (typ)
±0.5°C linearity (typ)
Stable with large capacitive loads
Specified −40°C to +125°C, operation to +150°C
Less than 50 µA quiescent current
Shutdown current 0.5 µA max
Low self-heating
Qualified for automotive applications

APPLICATIONS

Environmental control systems
Thermal protection
Industrial process control
Fire alarms
Power system monitors
CPU thermal management

GENERAL DESCRIPTION

The TMP35/TMP36/TMP37 are low voltage, precision centigrade temperature sensors. They provide a voltage output that is linearly proportional to the Celsius (centigrade) temperature. The TMP35/ TMP36/TMP37 do not require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C over the −40°C to +125°C temperature range.

The low output impedance of the TMP35/TMP36/TMP37 and its linear output and precise calibration simplify interfacing to temperature control circuitry and ADCs. All three devices are intended for single-supply operation from 2.7 V to 5.5 V maximum. The supply current runs well below 50 µA, providing very low self-heating—less than 0.1°C in still air. In addition, a shutdown function is provided to cut the supply current to less than 0.5 µA.

The TMP35 is functionally compatible with the LM35/LM45 and provides a 250 mV output at 25°C. The TMP35 reads temperatures from 10°C to 125°C. The TMP36 is specified from −40°C to +125°C, provides a 750 mV output at 25°C, and operates to 125°C from a single 2.7 V supply. The TMP36 is functionally compatible with the LM50. Both the TMP35 and TMP36 have an output scale factor of 10 mV/°C.

FUNCTIONAL BLOCK DIAGRAM

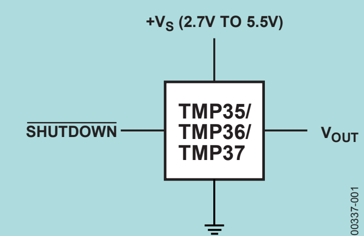


Figure 1.

PIN CONFIGURATIONS

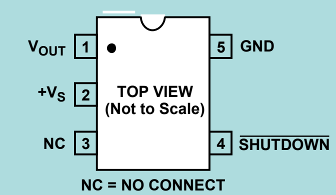


Figure 2. RJ-5 (SOT-23)

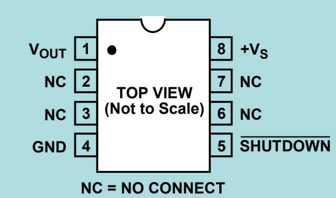


Figure 3. R-8 (SOIC_N)

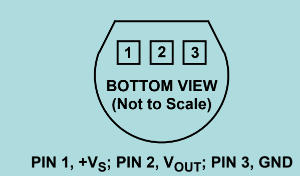


Figure 4. T-3 (TO-92)

The TMP37 is intended for applications over the range of 5°C to 100°C and provides an output scale factor of 20 mV/°C. The TMP37 provides a 500 mV output at 25°C. Operation extends to 150°C with reduced accuracy for all devices when operating from a 5 V supply.

The TMP35/TMP36/TMP37 are available in low cost 3-lead TO-92, 8-lead SOIC_N, and 5-lead SOT-23 surface-mount packages.

Know your Sensor

Temperature Sensor - TMP36

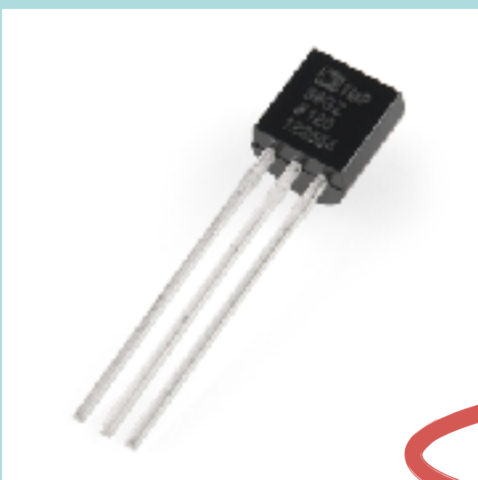


TABLE OF CONTENTS

Know your Sensor

Temperature Sensor - TMP36

ABSOLUTE MAXIMUM RATINGS

Table 2.

Parameter[1, 2]	**Rating**
Supply Voltage	7 V
Shutdown Pin	GND ≤ $\overline{\text{SHUTDOWN}}$ ≤ $+V_S$
Output Pin	GND ≤ V_{OUT} ≤ $+V_S$
Operating Temperature Range	−55°C to +150°C
Die Junction Temperature	175°C
Storage Temperature Range	−65°C to +160°C
IR Reflow Soldering	
Peak Temperature	220°C (0°C/5°C)
Time at Peak Temperature Range	10 sec to 20 sec
Ramp-Up Rate	3°C/sec
Ramp-Down Rate	−6°C/sec
Time 25°C to Peak Temperature	6 min
IR Reflow Soldering—Pb-Free Package	
Peak Temperature	260°C (0°C)
Time at Peak Temperature Range	20 sec to 40 sec
Ramp-Up Rate	3°C/sec
Ramp-Down Rate	−6°C/sec
Time 25°C to Peak Temperature	8 min

[1] Digital inputs are protected; however, permanent damage can occur on unprotected units from high energy electrostatic fields. Keep units in conductive foam or packaging at all times until ready to use. Use proper antistatic handling procedures.
[2] Remove power before inserting or removing units from their sockets.

Know yo

Temperature Sensor - TMP36

TYPICAL PERFORMANCE CHARACTERISTICS

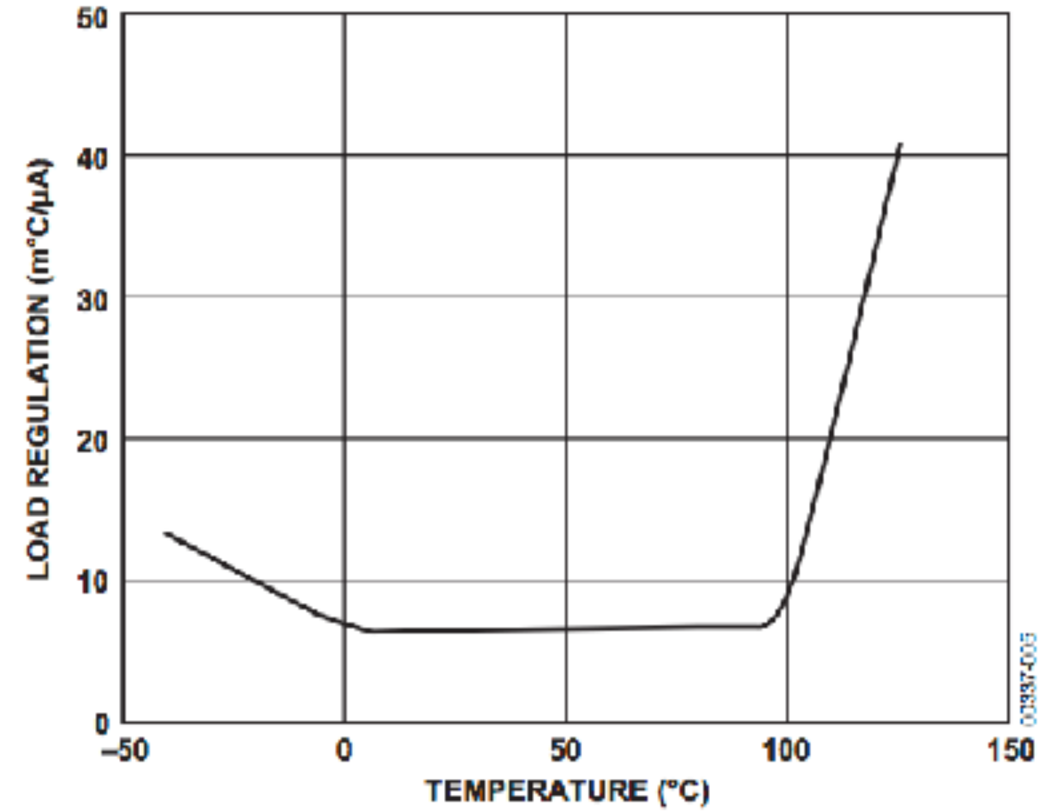


Figure 5. Load Regulation vs. Temperature (m°C/µA)

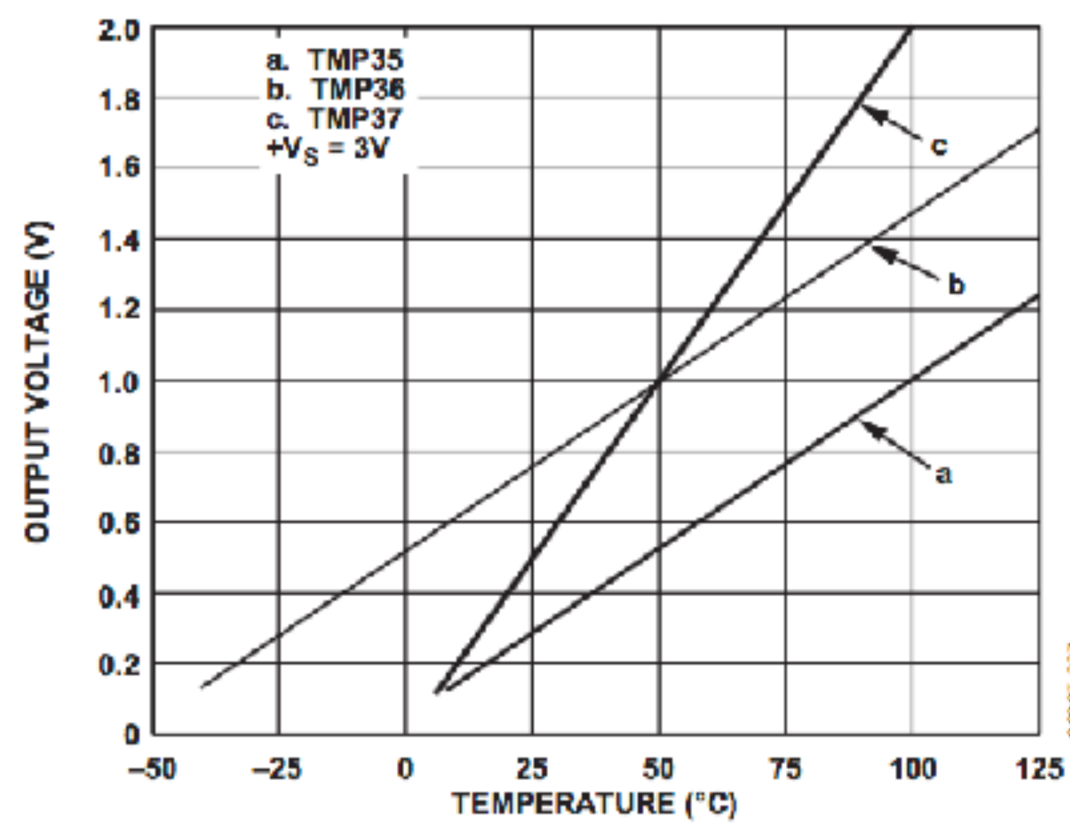


Figure 6. Output Voltage vs. Temperature

Know your Sensor

Temperature Sensor - TMP36

```
void loop()
{
  // Keep reading the sensor value so when we make an API
  // call to read its value, we have the latest one
  int reading = analogRead(tempPin);

  // The returned value from the device is going to be in the range from 0 to 4095
  // Calculate the voltage from the sensor reading
  double voltage = (reading * 3.3) / 4095.0;

  // Calculate the temperature and update our static variable
  temperature = (voltage - 0.5) * 100;

  // Now convert to Farenheight
  temperatureF = ((temperature * 9.0) / 5.0) + 32.0;
}
```

Know yo

Temperature Sensor - TMP36

TYPICAL PERFORMANCE CHARACTERISTICS

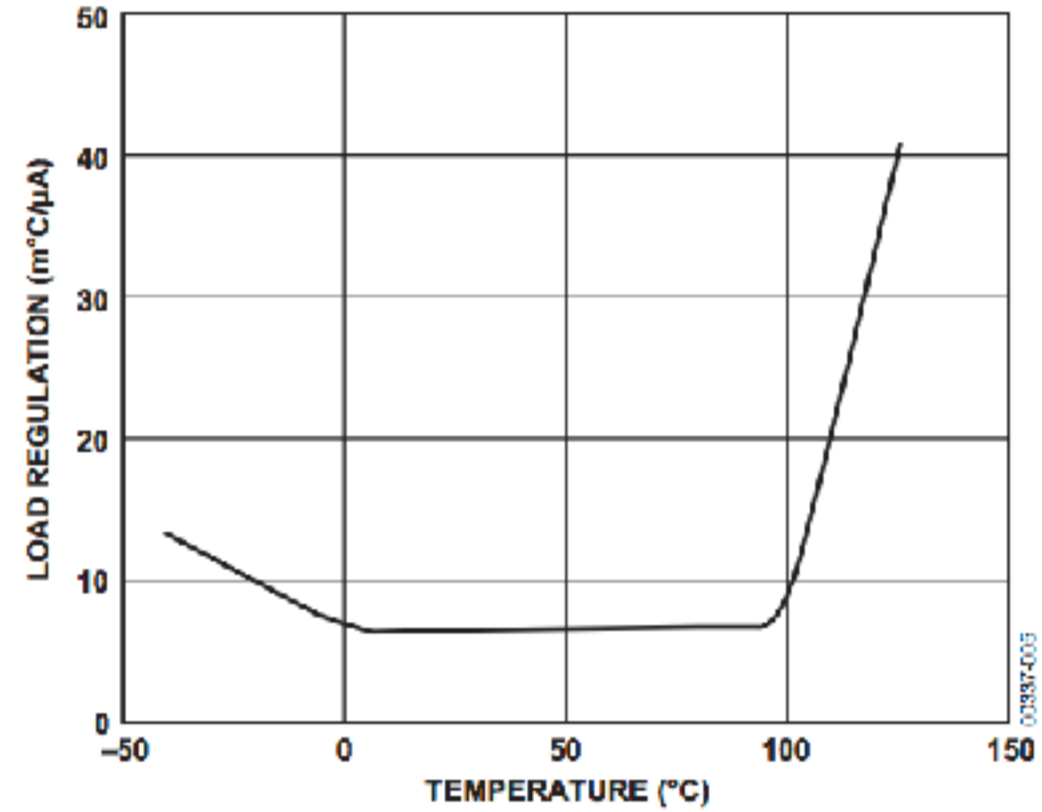


Figure 5. Load Regulation vs. Temperature (m°C/µA)

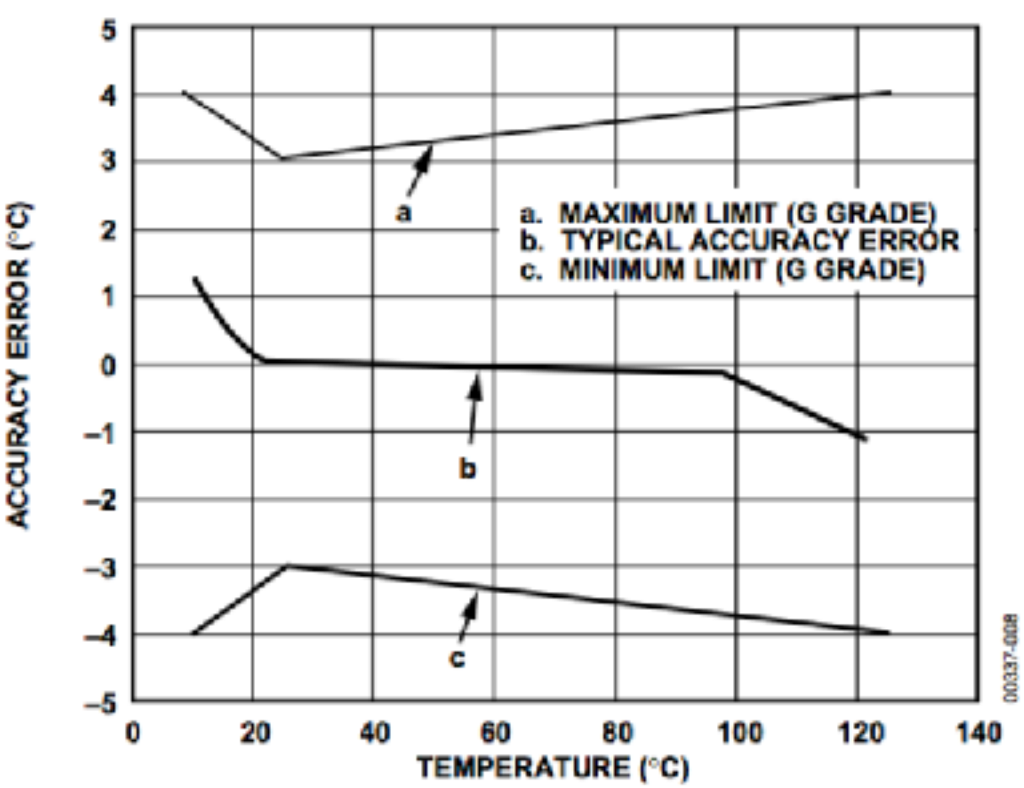


Figure 7. Accuracy Error vs. Temperature

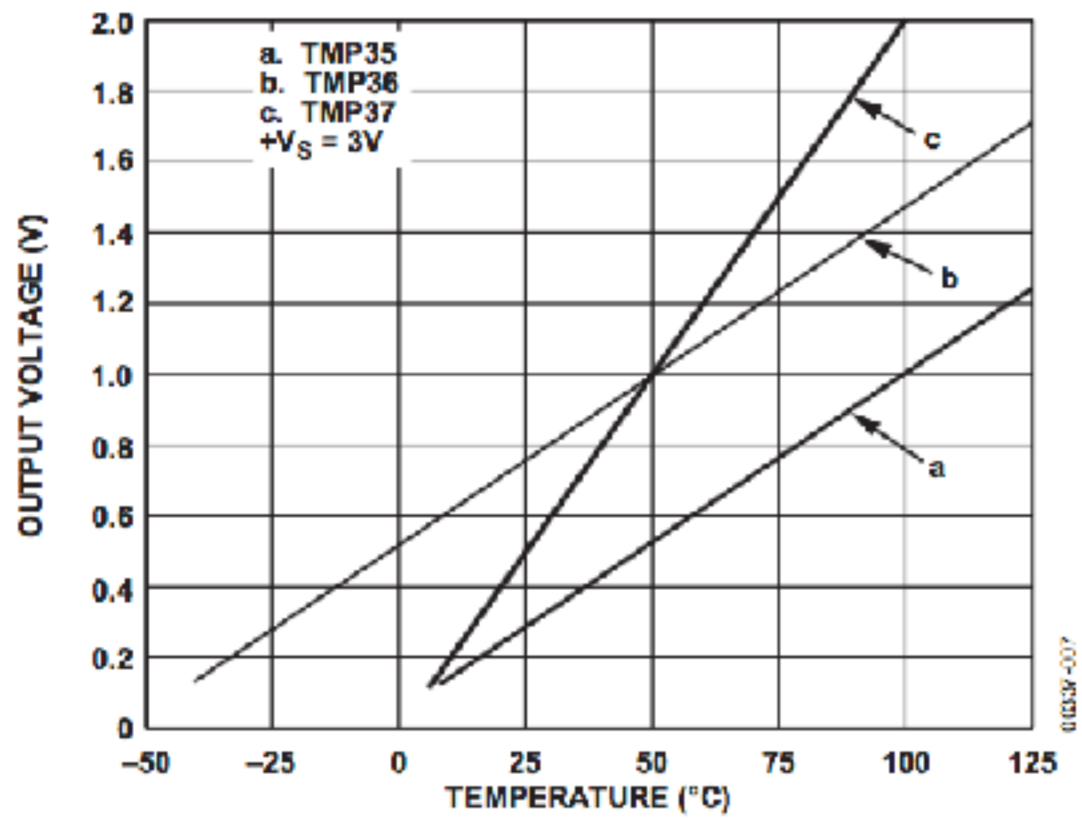


Figure 6. Output Voltage vs. Temperature

Know your Sensor

Temperature Sensor - TMP36

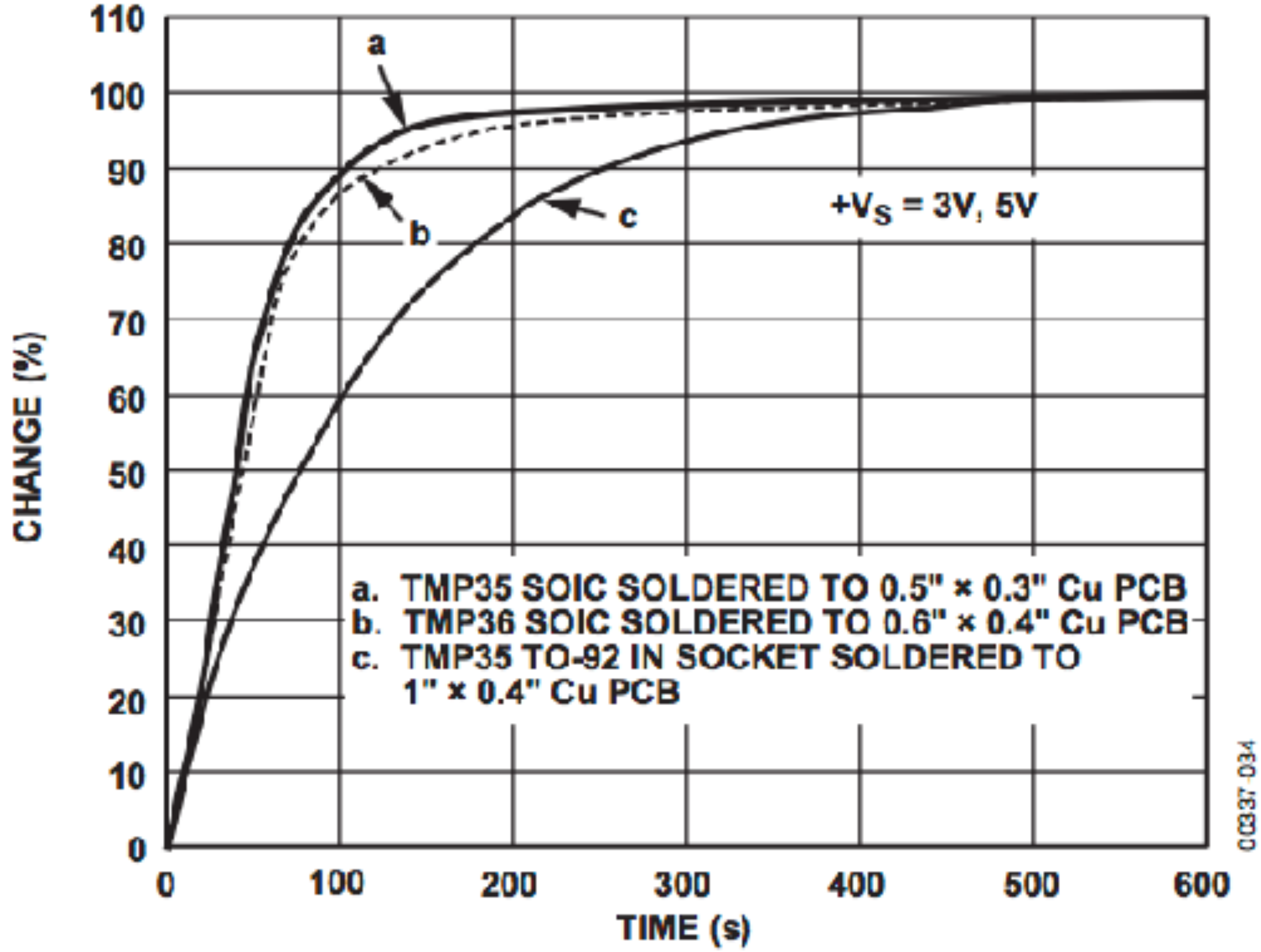
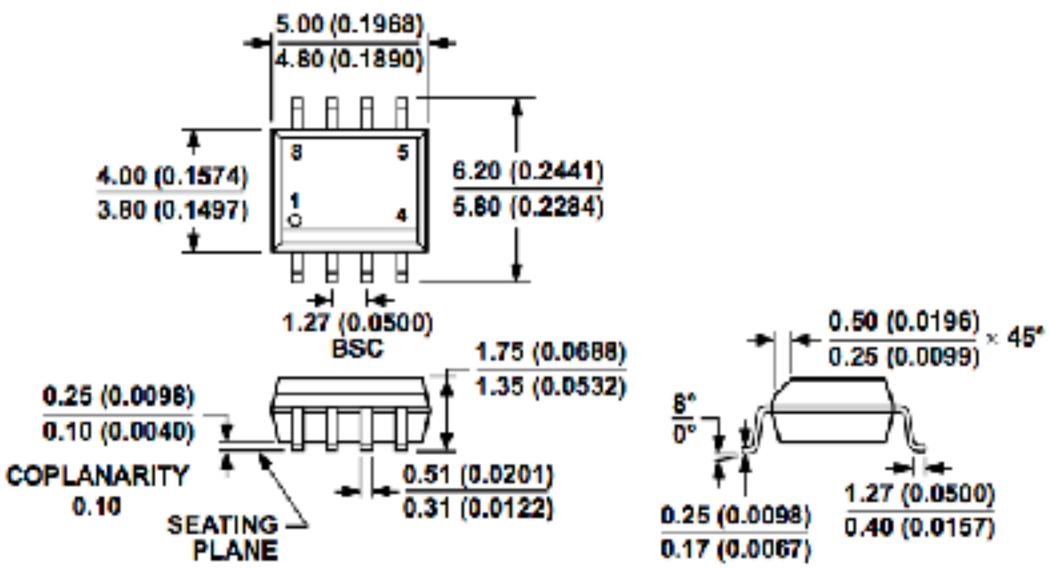


Figure 17. Thermal Response Time in Still Air

Know your Sensor

TMP35/TMP36/TMP37

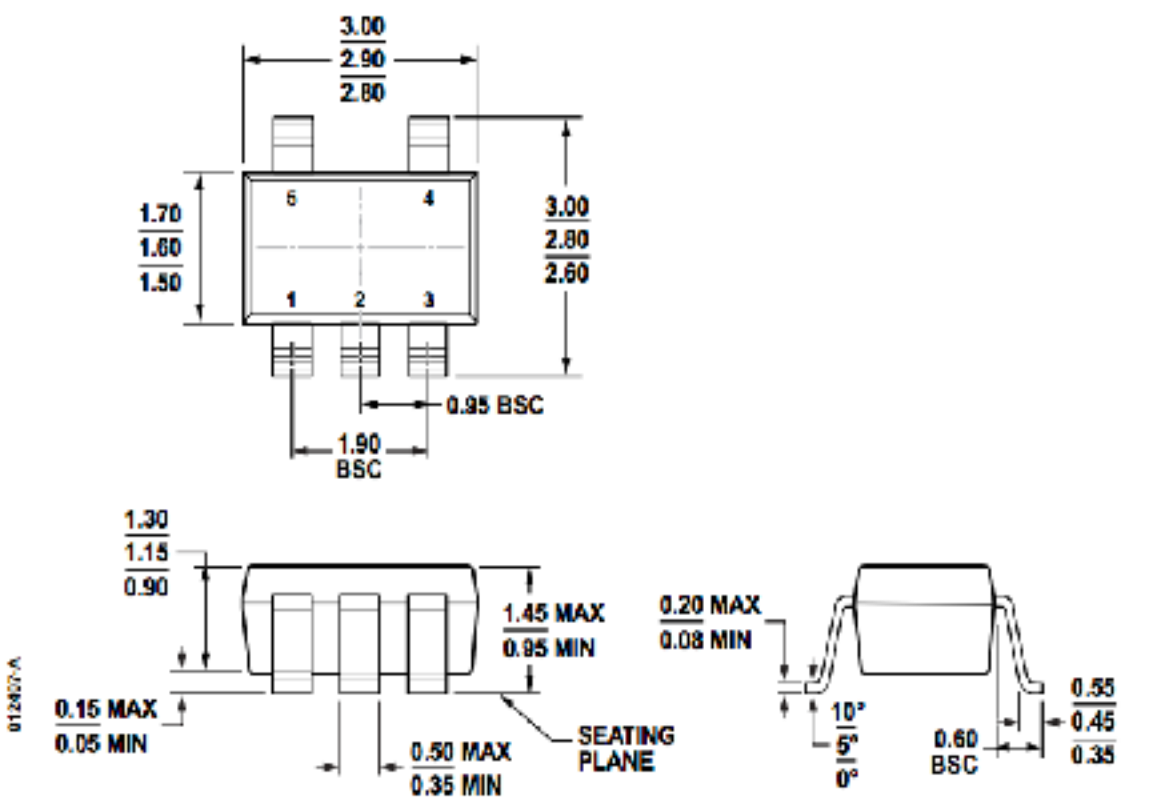
OUTLINE DIMENSIONS



COMPLIANT TO JEDEC STANDARDS MS-012-AA

CONTROLLING DIMENSIONS ARE IN MILLIMETERS; INCH DIMENSIONS (IN PARENTHESES) ARE ROUNDED-OFF MILLIMETER EQUIVALENTS FOR REFERENCE ONLY AND ARE NOT APPROPRIATE FOR USE IN DESIGN.

Figure 36. 8-Lead Standard Small Outline Package [SOIC_N]
Narrow Body
(R-8)
Dimensions shown in millimeters and (inches)



COMPLIANT TO JEDEC STANDARDS MO-178-AA

Figure 37. 5-Lead Small Outline Transistor Package [SOT-23]
(RJ-5)
Dimensions shown in millimeters

Datasheets

Only read what you need

90% of what's there you don't need. So don't read it

Skim it

Figure out what you need to know before looking

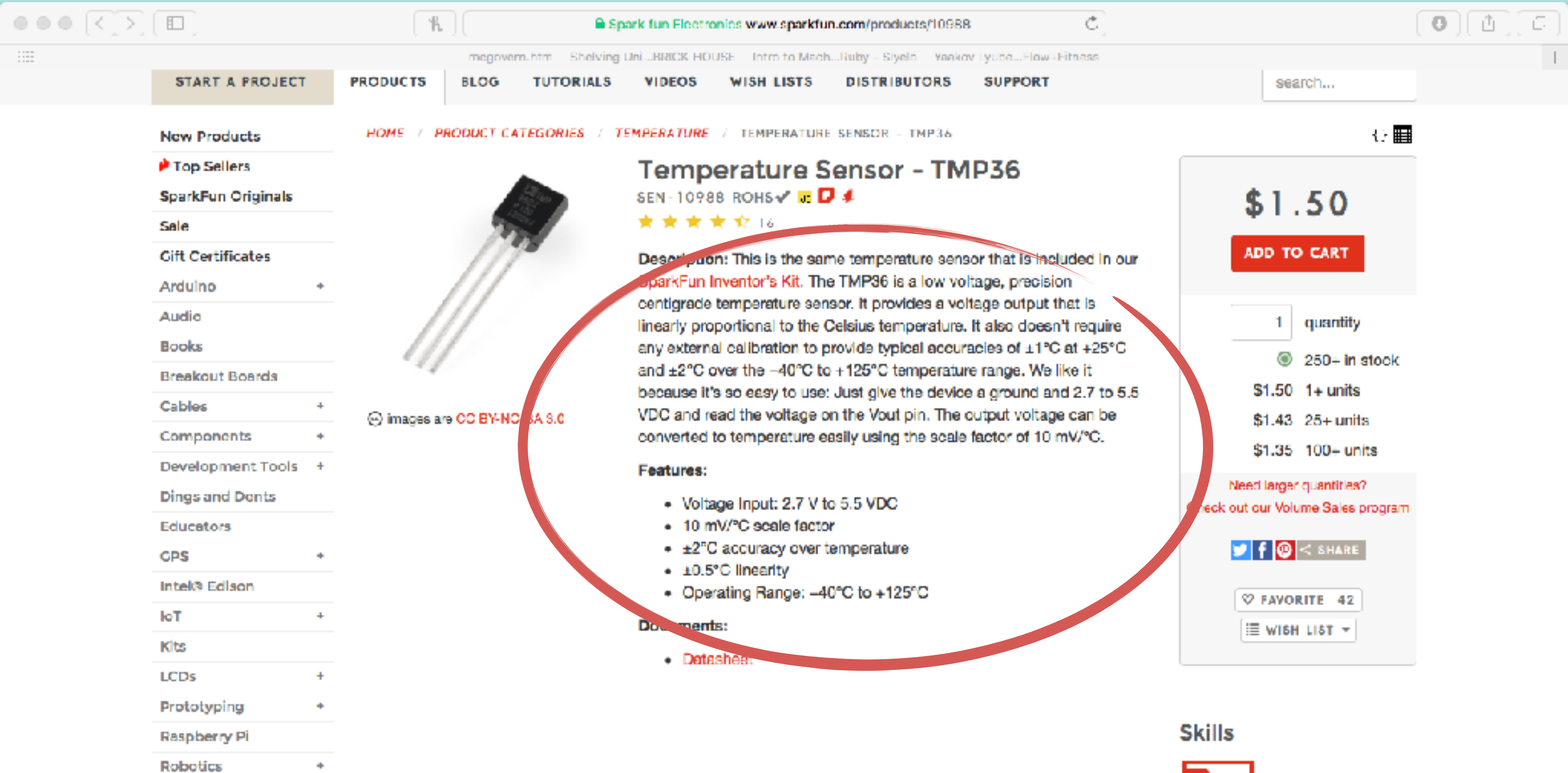
Check for special notes at the end

Check the version is current and the sheet matches your component

Better yet. Skip it. Search forums and guides online.

Datasheets

And don’t forget Sparkfun and other sites pull out most of what you need to know.



Know your Sensor

Now you know your sensor!

Yay!

Smoothing and Noise

Smoothing and Noise

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24
> 26
> 102
> 22
> 21
> 23
> 24
```

```
int val = analogRead( A0 );

Serial.println( val )

if( val > 100 ){
 digitalWrite( ledPin, HIGH);
 Particle.publish( “event-happened”);
}
```

Smoothing and Noise

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24
> 26
> 102
> 22
> 21
> 23
> 24
```

```
int val = analogRead( A0 );

Serial.println( val )

if( val > 100 ){
 digitalWrite( ledPin, HIGH);
 Particle.publish( “event-happened”);
}
```

Marvelous!

What’s the problem?

We’ve written code (announcing an event) that’s extremely sensitive to data quality.

Outliers and errors can easily trigger the outcome

This is especially problematic with low quality or imprecise sensors

e.g. PhotoCell

What’s the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24
> 26
> 102
> 22
> 21
> 23
> 24
```

Smoothing

Simply it shifts the values back into expected ranges.

If the change is sustained, the output of the smoothing algorithm will move towards the new upper bound. But it won’t jump there

What’s the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24
> 26
> 102
> 22
> 21
> 23
> 24
```

Windows Average

Most common approach.

You take the *n* past readings and average them to get the current *smoothed* value

What's the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24
> 26
> 102
> 22
> 21
> 23
> 24
```

```
23.6
```

Windows Average

Most common approach.

You take the *n* past readings and average them to get the current *smoothed* value

What's the fix

```
> 23
> 24
> 26
> 23      23.2
> 22
> 21
> 24
> 26
> 102
> 22
> 21
> 23
> 24
```

Windows Average

Most common approach.

You take the *n* past readings and average them to get the current *smoothed* value

What's the fix

```
> 23
> 24
> 26
> 23
> 22      23.2
> 21
> 24
> 26
> 102
> 22
> 21
> 23
> 24
```

Windows Average

Most common approach.

You take the *n* past readings and average them to get the current *smoothed* value

What’s the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24
> 26
> 102
> 22
> 21
> 23
> 24
```

```
23.2
```

Windows Average

Most common approach.

You take the *n* past readings and average them to get the current *smoothed* value

What’s the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24      39
> 26
> 102
> 22
> 21
> 23
> 24
```

Windows Average

Most common approach.

You take the *n* past readings and average them to get the current *smoothed* value

What’s the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24      39
> 26
> 102
> 22
> 21
> 23
> 24
```

Windows Average

Most common approach.

You take the *n* past readings and average them to get the current *smoothed* value

The bigger the window size the less sensitive it is to change

What's the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24     34.8
> 26
> 102
> 22
> 21
> 23
> 24
```

Windows Average

Most common approach.

You take the *n* past readings and average them to get the current *smoothed* value

The bigger the window size the less sensitive it is to change

What’s the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24
> 26
> 102
> 22
> 21
> 23
> 24
```

32.3

Windows Average

Most common approach.

You take the *n* past readings and average them to get the current *smoothed* value

The bigger the window size the less sensitive it is to change

What's the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24      32.3
> 26
> 102
> 22
> 21
> 23
> 24
```

But bigger windows = more computation. It slows down processing

i.e. don't do a window size of 2,000

The bigger the window size the less sensitive it is to change

What’s the fix

```
> 23
> 24
> 26
> 23
> 22
> 21
> 24     32.3
> 26
> 102
> 22
> 21
> 23
> 24
```

Generally speaking a windowed average is perfect for microcontrollers and most sensors.

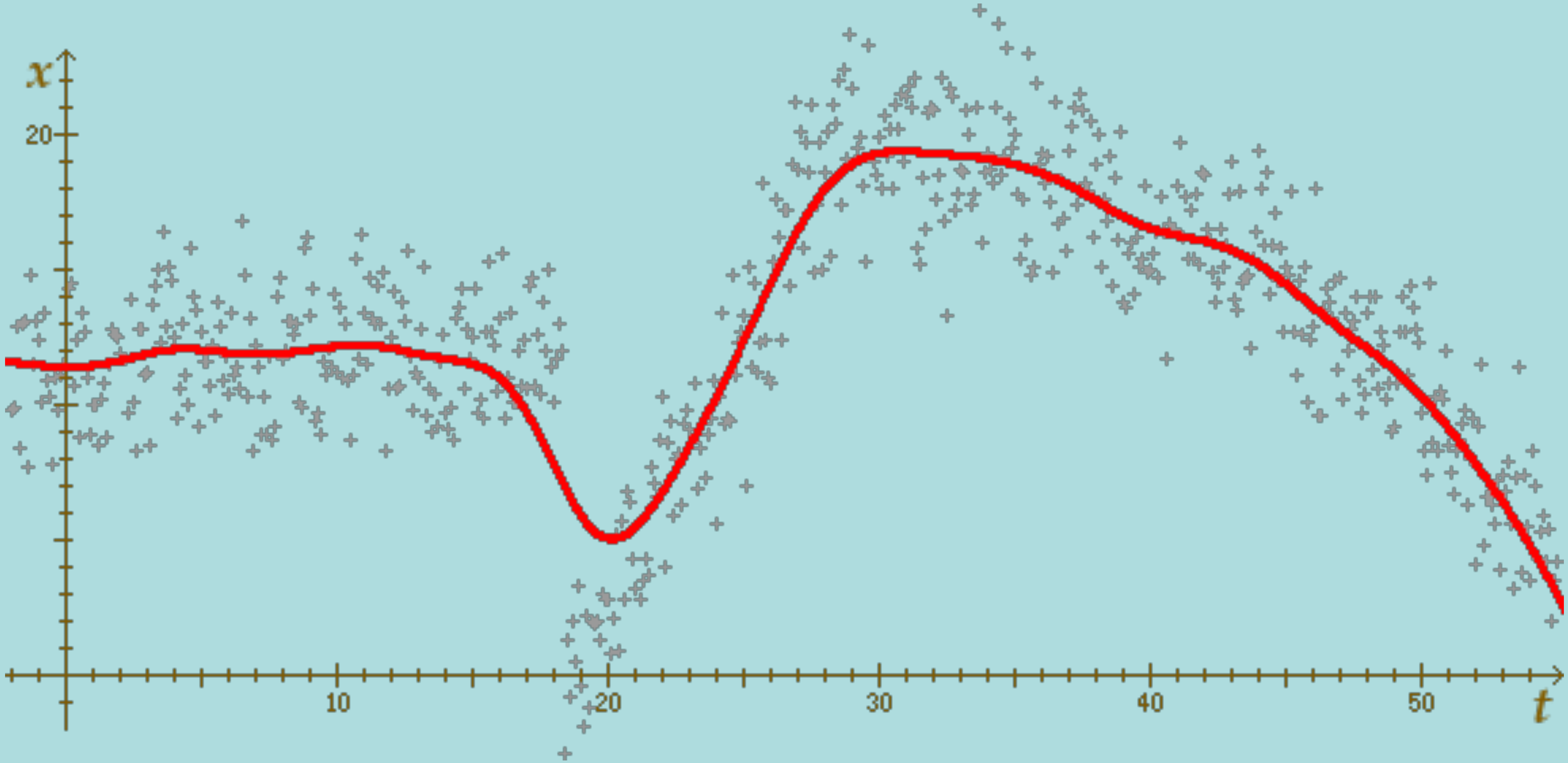
It’s fast, quick, simple to implement and improves data reliability massively

Simple Smoothing

```
const int numReadings = 10;

int readings[numReadings];      // the readings from the analog input
int readIndex = 0;              // the index of the current reading
int total = 0;                  // the running total
int average = 0;                // the average
int inputPin = A0;

void setup() {
 Serial.begin(9600);
 for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
  }
}

void loop() {
  total = total - readings[readIndex]; // subtract the last reading:
  readings[readIndex] = analogRead(inputPin); // read from the sensor:
  total = total + readings[readIndex]; // add the reading to the total:
  readIndex = readIndex + 1; // advance to the next position in the array:

  if (readIndex >= numReadings) {  // if we're at the end of the array...
   readIndex = 0; // ...wrap around to the beginning:
  }

  average = total / numReadings; // calculate the average:
  delay(50);        // delay in between reads for stability
}
```

Taken From: https://www.arduino.cc/en/tutorial/smoothing

Simple Smoothing



Taken From: https://math.stackexchange.com/questions/248843/removing-noise-when-the-signal-is-not-smooth

Other Implementations

Low Pass filters (see Brandon's tutorial!)

Simple: Implementation See: http://playground.arduino.cc/Main/DigitalSmooth

+ Lots more…

Exponential Moving Average (EMA); Savitzky Golay Filter (SGA); Ramer Douglas Peucker Algorithm (RDP); Kolmogorov Zurbenko Algorithm (KZA)

+ Kalman (dealing with imprecise data and translating it into outcomes; predictive, statistical inference based on unknown variables, typically used in robotics scenarios)

What to remember:

Noisy data is bad. It’ll happen with every sensor.

Some sensors are worse than others. Get to know which ones.

Don’t rely on a single point if you need to do precise things.

Algorithms to know about

General: Peak to Peak, Rate of Change, Windowed Averages,

Accelerometer: Acceleration, Sudden Impact Detection, Calculating Pitch, Yaw, Roll, Gesture analysis, simple to complex.

Sound: FFT,etc.

Algorithms to know about

There are lots of technical approaches to making your data useful.

Familiarize yourself with some possibilities.

Your job: Do some research

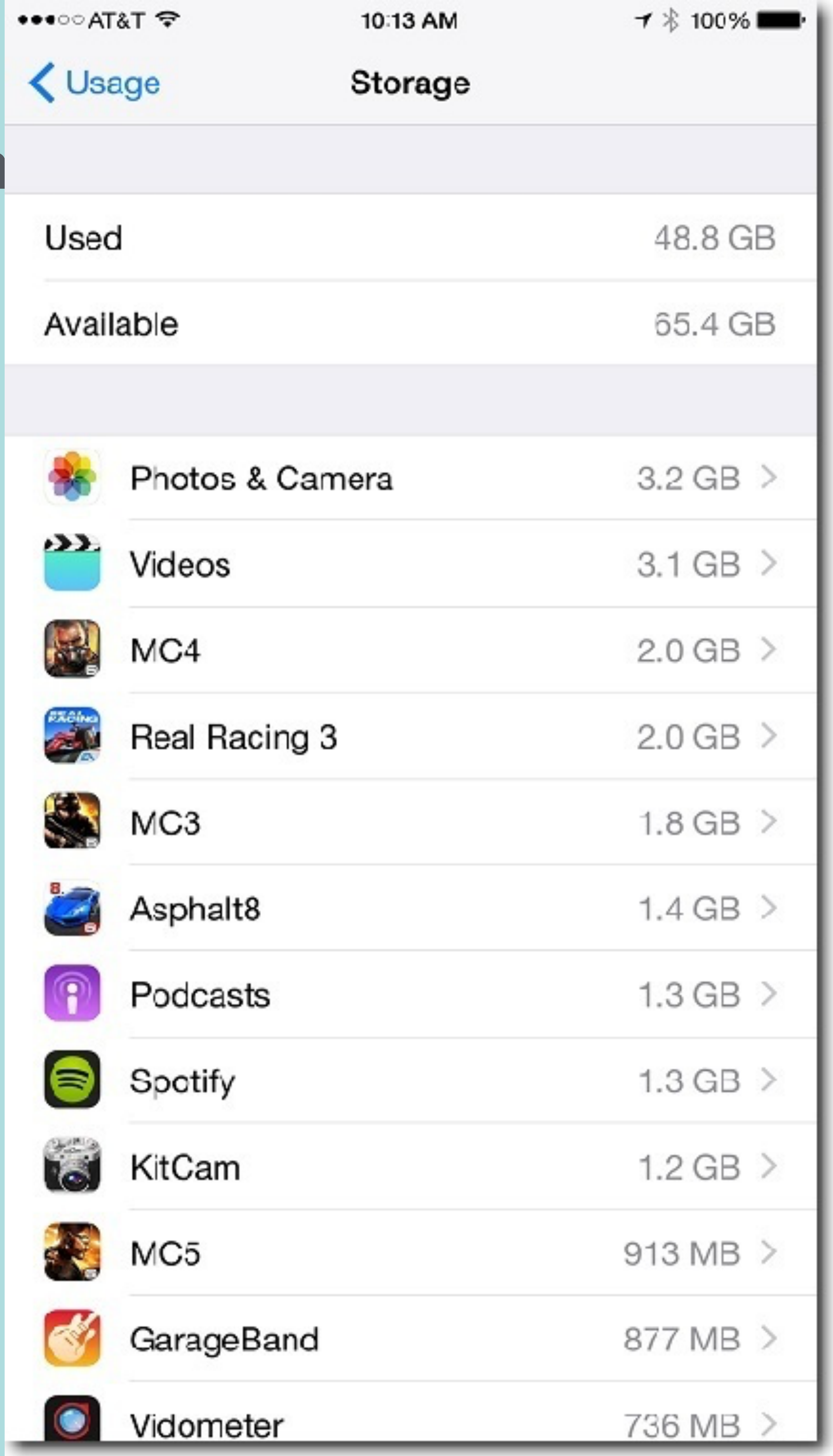
Arduino playground has years of contributions and most of the code works directly.

Avoid high tech unless you absolutely need to. Low tech often works better (for lots of reasons).

Limits of a Photon

- Tiny storage
  - 1MB flash, 128KB RAM
  - 512KB just for firmware/operating
- Exceptionally small programs ~128Kb
  - That includes libraries and other memory used
- ARM Cortex M3 micro-controller
  - System clock (120 MHz)

Limits of a Ph



This ain’t an iPhone

Limits of a Photon

We need to be smart about:

- Computation - we can’t do intensive/expensive tasks
- Performance - additional computation slows things down
- Information - we should only use what we need
- Storage - we can’t store much (even in arrays)

Limits of a Photon

What can we do or A lot with a little:

- Prefiltering data
  - Peak to Peak
- Being smart and efficient.
  - Do we really need heavy computation or gestures?
- Offloading to online services (Azure, Google Cloud)

Limits of a Photon

Reggies Problem: Recording a nights sleep.

If I sample every second:

28800000 readings

What can I do:

I can’t record all of the data (accelerometer every 5s)

Limits of a Photon

Reggies Problem: Recording a nights sleep.

If I sample every second:

- 28800000 readings
- 3 axes of accelerometer.

w/ 4 bytes per int = 34 MB raw

We could sample less often; but we loose precision.

Limits of a Photon

But we’re really only trying to find the right time range (in 20 minutes-ish blocks) to wake someone.

We can keep precision on new data

And store old data in simpler ways.

Limits of a Photon

Strategy:

Hold readings for past minute in a buffer

Get cumulative average for last minute,

store this on the device

x 60000 less data.

Limits of a Photon

Strategy:

Hold readings for past minute in a buffer

Get cumulative average for last minute,

store this on the device

x 60000 less data.

Limits of a Photon

Strategy:

Getting better.

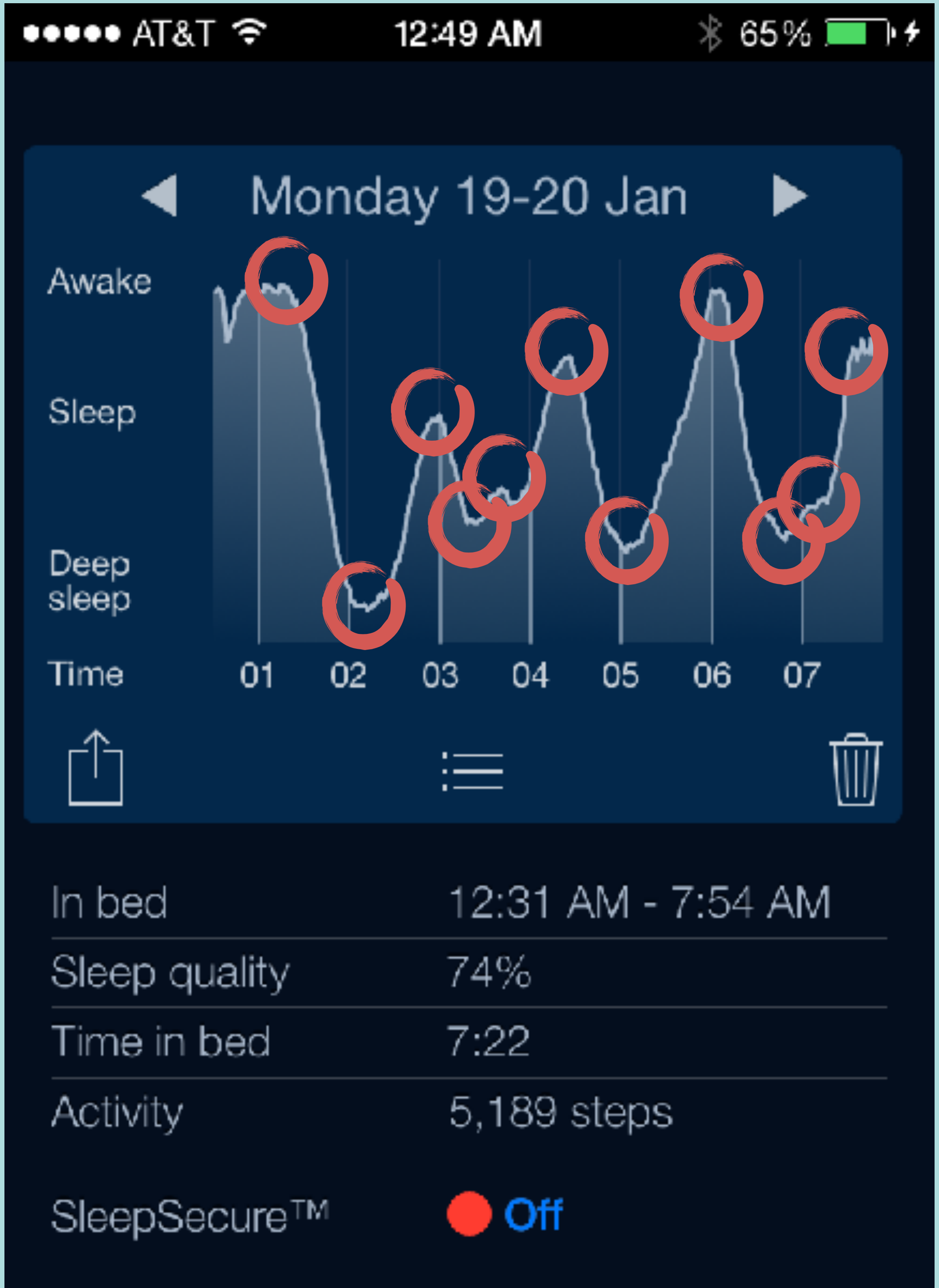
We're really only interested in changes and transitions between states

Limits of a Photon

Strategy:

Getting better.

We're really only interested in changes and transitions between states



Limits of a Photon

Strategy:

Getting better.

We're really only interested in changes and transitions between states

So why hold the raw movement?

Limits of a Photon

Strategy:

We're really only interested in

1. **Peaks and Troughs**

2. **The rate of change and time to change between them**

This tells us sleep quality

So calculate as we go

Limits of a Photon

```
// A binary search based function that returns index of a peak element
int findPeakUtil(int arr[], int low, int high, int n)
{
    // Find index of middle element
    int mid = low + (high - low)/2;  /* (low + high)/2 */

    // Compare middle element with its neighbours (if neighbours exist)
    if ((mid == 0 || arr[mid-1] <= arr[mid]) &&
            (mid == n-1 || arr[mid+1] <= arr[mid]))
        return mid;

    // If middle element is not peak and its left neighbour is greater
    // than it, then left half must have a peak element
    else if (mid > 0 && arr[mid-1] > arr[mid])
        return findPeakUtil(arr, low, (mid -1), n);

    // If middle element is not peak and its right neighbour is greater
    // than it, then right half must have a peak element
    else return findPeakUtil(arr, (mid + 1), high, n);
}

// A wrapper over recursive function findPeakUtil()
int findPeak(int arr[], int n)
{
    return findPeakUtil(arr, 0, n-1, n);
}
```

Limits of a Photon

```
String info[100];

info[0] = “Peak,891,123445554”
info[1] = “Trough,91,123545554”
```

Limits of a Photon

Greatly simplify, without loosing what we need

Limits of a Photon

What can we do or A lot with a little:

- Reducing data / sampling
- Prefiltering data
  - Peak to Peak
- Being smart and efficient.
  - Do we really need heavy computation or gestures?
- Offloading to online services (Azure, Google Cloud)