ThermalMonitor
User's Guide

Document Version 1.3
June 15, 2015

Shane Whalen
System Software Engineering, Firmware
shane_whalen@apple.com

Contents

List of Figures

List of Tables

1 Introduction

1.1 What is ThermalMonitor?

ThermalMonitor is an idle task in EFI that allows for temperature tracking and DUT over-temperature shutdowns. Its goals are to eliminate damage due to overheating and to allow post-processing of collected temperature data over the factory lifecycle of a DUT.

1.2 Motivation

During the 2014 product cycle it was found that DUTs had been damaged during the course of production testing due to overheating while in EFI. EFI does not have CLTM (Closed Loop Thermal Monitor) and did not detect the DUT temperatures rising beyond their tolerances. The DUTs were heating due to being stacked inside improperly ventilated trays while in a high power state. Data analysis concluded that devices from previous years had also seen overheating, but without the obvious physical manifestations that had occurred during 2014.

1.3 Document Objective

This document is generally intended to be an all-around reference for the consumers and producers of ThermalMonitor and its usage. The goal is to enable consumers to analyze and diagnose thermal issues in the system using ThermalMonitor. Although this document will not cover technical details, the document will explain the design and shed light on the considerations required to maintain or enhance the software.

2 Monitoring

2.1 Idle Task

The DUT is considered idle after 500ms of inactivity (no commands running); otherwise, the DUT is considered busy.

There is no activity in ThermalMonitor while the DUT is busy. In the idle state, each available sensor will be polled at a fixed interval. When activity occurs on the DUT the ThermalMonitor will stop execution.

It is possible to disable the idle task through DiagsShell. See section 3 *Using ThermalMonitor*.

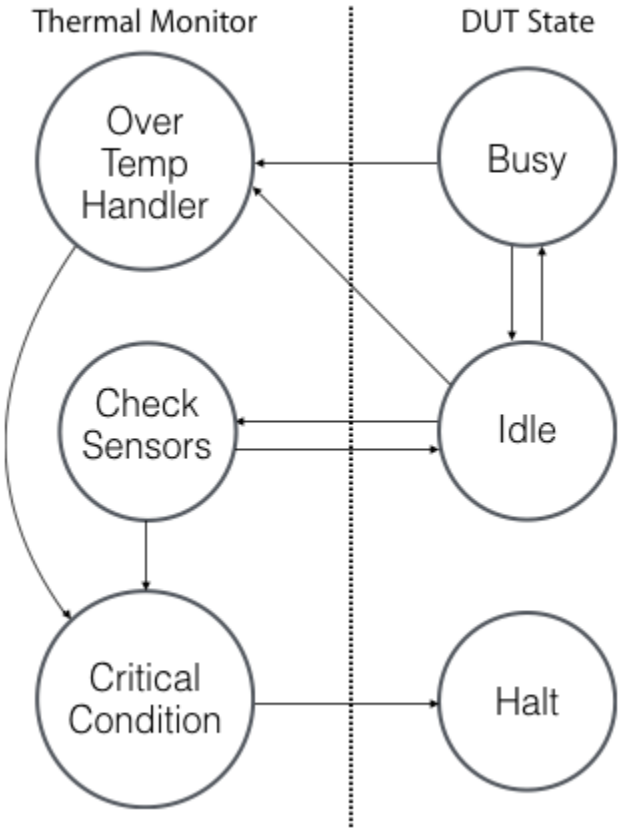

Figure 1: *ThermalMonitor Block Diagram*

2.2 Sensor Behavior

Each sensor can have one of three modes:

Data Collection — There is no temperature limit associated. Temperature data will be collected at the preprogrammed interval.

Limit Enabled — Same as Data Collection, but the DUT will shut down if the temperature exceeds the configurable limit while the DUT is idle.

Critical Limit — Same as Limit Enabled, but the DUT will shut down if the temperature limit is exceeded at any point.

- This is accomplished using temperature interrupts associated with each sensor. The interrupts are configured to trigger only when an over-temperature event occurs so as not to bother busy, normal temperature, DUT.
- The interrupt on over-temperature cannot be disabled, but the limit can be modified. See section 3.2.7 *Change a Sensor Limit*.
- Some sensors are not capable of this configuration and cannot be configured to have Critical Limits.

Each sensor's mode is predefined in the EFI binary and cannot be changed during runtime.

2.3 Shutting Down the DUT

When a limit is exceeded for either a Limit Enabled or a Critical Limit sensor the DUT is shutdown in order to prevent damage due to overheating.

Before the DUT can be shutdown, information about the exceeded limit is recorded. Due to software limitations, EFI is incapable of saving this information to a file during the shutdown process. Instead, the information is stored in the PMU scratch space, which persists across reboot cycles.

Once the event information is stored in the PMU, the DUT is put into its hibernate state. Hibernate (S2R) was chosen over Standby (OFF) to allow DUTs to remain in a low power state while they are attached to the dock connector.

The information stored about the limit failure can be obtained at a later time. See section 4.4 *ThermalMonitor Shutdown*.

3 Using ThermalMonitor

3.1 Data Storage

Data collected in ThermalMonitor is stored in RAM until it receives notification to flush to file via internal EFI events. This will occur by default during shutdown or reset. The data can also be flushed manually by using the *event* command. See section 3.2.5 *Saving Data to a File*. If the internal buffer is full the data will also be flushed. The data will be appended to the end of the log file and the RAM buffer is reset and emptied.

The contents of the file are in the same CSV format mentioned in section 3.2.4 *Display Collected Temperatures*.

The file is located on the Log partition. If the Log partition does not exist no data will persist though reboot. The path to the data file is:

```
Logs:\MobileMediaFactoryLogs\LogCollector\FactoryDebug\temperature_runtime.csv
```

3.2 Commands

ThermalMonitor also exposes a subset of its functionality through DiagsShell.

3.2.1. Print Command Line Help

```
help thermalmonitor
```

Print a list of all supported command line options. Not all option combinations are valid.

3.2.2. Disable ThermalMonitor

```
thermalmonitor --off
```

Disable the ThermalMonitor idle task. No temperature data will be collected unless manually requested (see section 3.2.6 *Check Temperatures Immediately*). While off the DUT can only will only be shutdown if a Critical Limit sensor detects an over-temperature event. See section 2.2 *Sensor Behavior*.

3.2.3. Enable ThermalMonitor

```
thermalmonitor --on
```

Enable the ThermalMonitor idle task. This is the default state of ThermalMonitor when EFI is started.

3.2.4. Display Collected Temperatures

```
thermalmonitor --dump
```

Print all temperatures collected since the last time data was flushed to a file. The contents of the data buffer will not be changed and data will not be stored to a file. The output will be in CSV format:

<*unix_time*>, <*id*>, <*dev_name*>, <*temperature*>

Data previously written to the log file will not be show. However, the data can be displayed using the cat command.

```
:-) cat Logs:\MobileMediaFactoryLogs\LogCollector\FactoryDebug\temperature_runtime.csv
1420831170.793s, 0x00, soc, THERMAL0, 27.60
1420831170.793s, 0x01, soc, THERMAL1, 29.00
1420831170.793s, 0x02, soc, CCC_THERMAL0, 27.60
1420831170.793s, 0x03, soc, CCC_THERMAL1, 27.39
1420831170.794s, 0x04, soc, CCC_THERMAL2, 27.39
1420831170.796s, 0x28, pmu, TJINT, 32.63
1420831170.827s, 0x29, pmu, TDEV1, 24.70
1420831170.831s, 0x2A, pmu, TDEV2, 25.46
1420831170.835s, 0x2B, pmu, TDEV3, 25.21
1420831170.839s, 0x2C, pmu, TDEV4, 30.60
1420831194.312s, wrote at: 2015.1.9.19.19.54
```

3.2.5. Saving Data to a File

⌨ `event -s media-sync`

The data that was previously in RAM (displayed via the *--dump* option) will now be store to a file. This is done automatically during shutdown or reboot.

3.2.6. Check Temperatures Immediately

⌨ `thermalmonitor --check`

Check every available sensor's temperature. If an enabled limit is exceeded, the DUT will shut-down. Otherwise the command will output all of the newly collected temperatures in the same format as the *--dump* option.

3.2.7. Change a Sensor Limit

⌨ `thermalmonitor --dev <dev_name> --name <sensor_name> --setlimit`

Must be used in conjunction with *--dev* and *--name* in order to select the limit to update. The below command will set the limit for pmu.TDEV3 to 50. This will only affect Critical Limit and Limit Enabled sensors.

```
:-) thermalmonitor --dev pmu --name TDEV3 --setlimit 50
```

3.2.8. Display a Sensor Limit

⌨ `thermalmonitor --dev <dev_name> --name <sensor_name> --getlimit`

Must be used in conjunction with *--dev* and *--name* in order to select the limit to update. The below command will get the limit for pmu.TDEV3.

```
:-) thermalmonitor --dev pmu --name TDEV3 --getlimit
Limit TDEV3: 50.000000
```

3.2.9. List Settings

```
thermalmonitor --list
```

Display a list of available sensors and their settings. The following data will be displayed for each sensor:

```
:-) thermalmonitor --list
<dev_name>.<sensor_name>:
    SensorId: (<id>)
    Limit: <temperature>
    Period(ms): <period>
    Critical: <Yes|No>
    LimitEnabled: <Yes|No>
```

Keyword	Description
<*dev_name*>	The name of the device the sensor is connect to. E.g. "pmu" or "soc".
<*sensor_name*>	The name of the sensor. E.g. "TDEV3" or "CCC_THERMAL1".
<*id*>	A unique identifier for the sensor. It is generated while booting into EFI. The identifier will stay constant across the reboots of the same version of EFI. It can be changed if a new sensor gets built into the EFI binary. See section 4 *Post-Processing* for more information on its use.
<*temperature*>	User modifiable upper limit of normal operating temperature.
<*period*>	The minimum time between readings when the DUT is idle.
Critical <*Yes|No*>	"Yes" indicates that the sensor has the ability to shutdown the DUT on over-temperature events at any point. "No" indicates the DUT must be at idle in order to handle over-temperature events.
LimitEnabled <*Yes/No*>	"Yes" indicates that an over-temperature event will shutdown the DUT. "No" indicates that ThermalMonitor will only collect temperature data from the sensor.

Table 1: *Sensor Settings*

3.3 Updating Default Limits

The default limits are compiled into the EFI binary and only modifiable by EFI engineers. The limits should be disabled at the beginning of a product cycle. Once enough data has been collected the limits should be given to EFI engineers by the System EE team.

4 Post-Processing

4.1 Temperature Data

Min and max temperature data along with the raw data is uploaded to PDCA at each restore and log collection station. The raw data can be futher analyzed by a tool called *AnalyzeTemps.py*. If the DUT isn't going through normal factory process the raw temperature data can be transfered to the host via rsync, dfufile, usbfs, etc.

4.2 Using AnalyzeTemps

The AnalyzeTemps script can be found as an attachment in the following radar: <rdar://problem/19294938> ThermalMonitor - Scripts and Tools

4.2.1. Command Line Summary

Below is a list of available options when using the tool.

```
$ python AnalyzeTemps.py --help
Usage: AnalyzeTemps.py [options]

Options:
  -h, --help            show this help message and exit
  -f FILE, --file=FILE  File containing the temperature data
  -o, --output          print the temperature data
  -p, --plot            plot the temperature data
  -i, --info            Output info about the data
  -d DEV, --device_id=DEV
                        Only use data from this device id
  -t TYPE1,TYPE2,..., --device_type=TYPE1,TYPE2,...
                        Only use data from these device types. pmu, soc, etc
  -n NAME1,NAME2,..., --sensor_name=NAME1,NAME2,...
                        Only use data from these sensor names
  -s START_TIME, --start_time=START_TIME
                        Only use data from this start time
  -e END_TIME, --end_time=END_TIME
                        Only use data up to this end time
```

4.2.2. Data Summary

```
python AnalyzeTemps.py -f temperature_runtime_example.csv -i
```

This will print a summary of each sensor recorded in the file. Below is a snippet from the output of the command.

```
$ python AnalyzeTemps.py -f temperature_runtime_example.csv -i
```

```
pmu.TDEV3:
    Num samples: 141
    First sample time: 1418399046.660 (2014-12-12 10:44:06)
    Last sample time: 1418760241.201 (2014-12-16 15:04:01)
    Min temperature: 23.94
    Min temperature time: 1418753746.572 (2014-12-16 13:15:46)
    Max temperature: 46.64
    Max temperature time: 1418760241.201 (2014-12-16 15:04:01)
```

The temperatures will be displayed in the units they were collected in. Units are not stored in the file, so they are not displayed. The timestamps are dependant on the programmed value in the RTC. Usually this is programmed to GMT.

4.2.3. Limiting the Results

The results can be further narrowed down via time, device ID, device name, or sensor name.

```
python AnalyzeTemps.py -f FILE -o -t pmu -n TDEV4 -s 1418760127
```

This command filters the output to be from sensors attached to the PMU named TDEV4. The results are further reduced by filtering out all the data points before the specified start time. The *--start-time* option should be in Unix time and can be translated into human readable format via an online tool[1].

[1] http://www.epochconverter.com

4.2.4. Plotting Data

The data can also be plotted using the same tool. The plot will display on the host in a separate window from the terminal.

```
python AnalyzeTemps.py -f temperature_runtime_example.csv -p
```

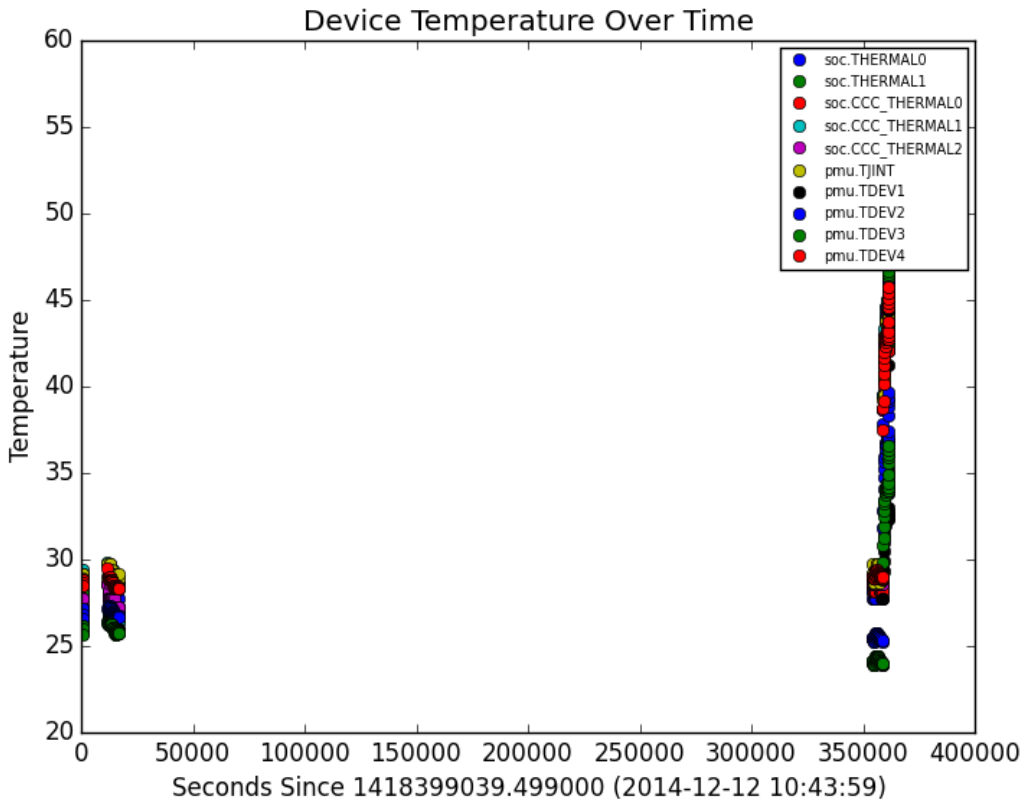

Figure 2: *Basic Temperature Plot*

4.2.5. Plotting Filtered Results

The plotting functionality also works with the filtering options.

```
python AnalyzeTemps.py -f temperature_runtime_example.csv -p -t pmu -s 1418750040
```

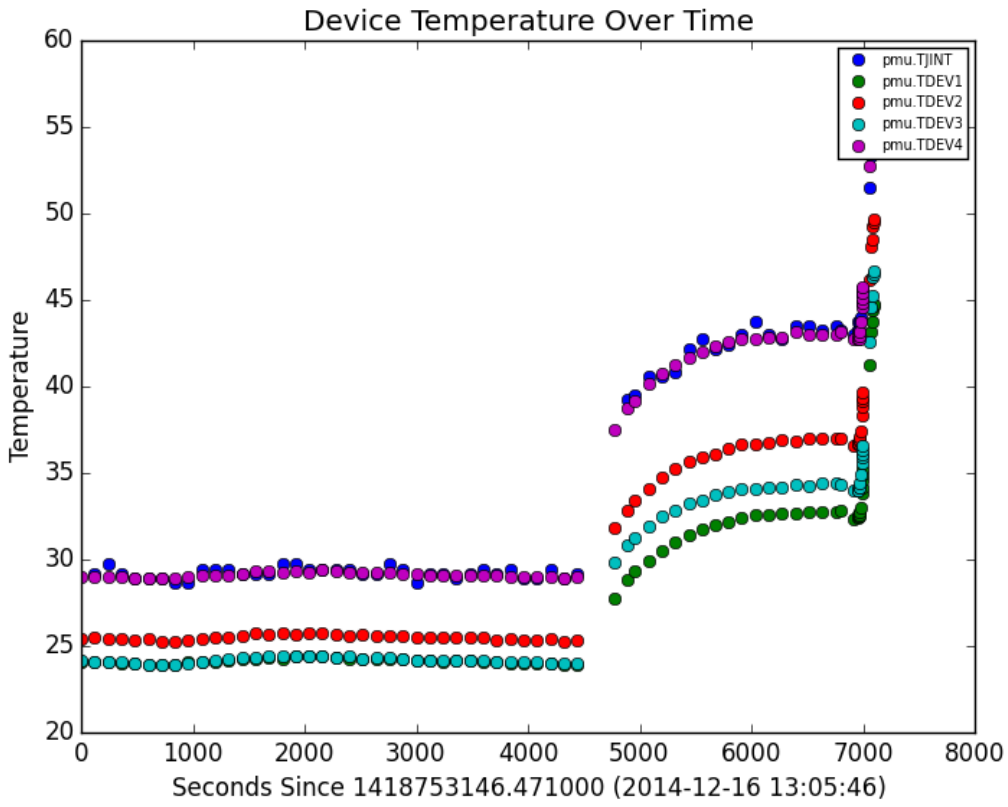

Figure 3: *Filtered Temperature Plot*

4.3 PDCA Parametric Data

The PDCA keys are in the following format:

<MAX_TEMP|MIN_TEMP>.<dev_name>.<sensor_name>

For example, the maximum GasGauge termperature can be found via key:

MAX_TEMP.battery.GasGauge

Min and max data are reset after each restore. The parametric data is uploaded at each restore station so no data is lost. The data is uploaded using the station ID of the corresponding restore or log collection station.

4.4 ThermalMonitor Shutdown

As mentioned in section 2.3 *Shutting Down the DUT*, the ThermalMonitor task will save critical data about the sensor that exceeded the temperature threshold before the DUT is shutdown. During a subsequent entry into EFI, this information is moved from the PMU to a file. The file is located in the following location:

```
Logs:\MobileMediaFactoryLogs\LogCollector\FactoryDebug\shutdown.log
```

If an over-temperature shutdown occurs, a BootStageDebugThermal entry will be appended to the end of the file with the following format:

```
Type: BootStageDebugThermal
    Time: 2014-12-15 19:04:17
    CommitId: CA30
    SensorId: 0x2C
    TempC: 23.25
    LimitC: 20.00
```

5 Future Work

5.1 Radars

<rdar://problem/19179025> Convert Sochot to be a thermal event with ThermalMonitor
SocHot is a separate mechanism in EFI and should be merged with the ThermalMonitor

<rdar://problem/19432841> Make ThermalMonitor post processing scripts available in the non-ui SDK
Make obtaining the post processing script easier.