Improvisation and demonstration of Linux thermal framework for multiple temperature sensors

Adithya K V and Tauseef Nomani
Need of Thermal Management Unit
Overview of TMU
Overview on Thermal framework
Thermal Management in Linux Kernel
Thermal device structure and functions
Pseudocode of Conventional TMU driver
Demo of interface from user space for Thermal Framework
TMU in Complex SoC
Limitation in Conventional driver for TMU in Complex SoC
Pseudocode of Complex SoC TMU driver
Scope for improvisation in thermal framework
SoC temperature increases during High computation and High frequency operation or during High Load.

High temperature of SoC results in:
- SoC poor performance
- SoC malfunction
- SoC permanent damage

Solution:
- **Thermal throttling**: Clock speed will be reduced and performance will be limited to reduce the heat buildup.
- **Thermal cooling**: Switch on the fan or any other cooling device.
- **Thermal tripping**: Indicate PMU or voltage regulator to cut-off the power supply to SoC.

Above mentioned solutions can be achieved using Thermal Management Unit (TMU).
Thermal Management Unit (TMU)

- TMU have a Controller and a Sensor integrated with in.
- Each block of SoC can have separate TMU placed in it, if temperature need to be monitored.
- Controller configures the temperature sensor and initiates sensing of temperature.
- TMU can be configured with different Temperature Threshold Levels.
- TMU generates Interrupt when sensed temperature crosses these Threshold Levels.
- TMU will have separate Threshold level and Interrupts for Thermal throttling and Thermal tripping.
- Some of vendors provide Emulation Mode support in their TMU.
Overview on Thermal framework

- Thermal zone device -> TMU + Temperature Sensor
- Cooling devices -> Fan or any other device
- Exposes thermal zone devices and cooling devices to the user space.
- These devices has to be registered with thermal framework.
- Registered devices becomes part of thermal management and made available to the user space
- User space application can make decisions based on current temperature and threshold temperatures
Thermal device ops Structures

- `struct thermal_zone_device_ops`

Commonly used function pointers

- `bind`: binds the thermal zone device with a thermal cooling device.
- `unbind`: unbinds thermal zones from thermal cooling device.
- `get_temp`: reads the sensor temperature.
- `set_trips`: sets the trip temperature window.
- `change_mode`: switches thermal management between kernel and user space.
- `get_trip_temp`: get trip temp threshold above which trip interrupt will be triggered.
- `set_trip_temp`: change trip temp threshold.
- `set_emul_temp`: set the emulation temperature
struct thermal_zone_of_device_ops (device tree)

Below are basic function pointers

- get_temp: reads the sensor temperature.
- get_trend: calculates rate of change of temperature.
- set_trips: sets the trip temperature window.
- set_emul_temp: set the emulation temperature.
- set_trip_temp: change trip temp threshold.
Cooling device ops Structures

- **struct thermal_cooling_device_ops**

  Below are basic function pointers

- **get_max_state**: Max possible state of cooling devices
- **get_cur_state**: read current state of cooling device.
- **set_cur_state**: sets current state of cooling device.
- **getRequested_power**: calculates the power requested by cooling device.
- **state2power**: Calculate the power consumption based on cooling device state.
- **power2state**: Calculate the stage based on power consumption.

```c
struct thermal_cooling_device_ops {
  int (*get_max_state)(struct thermal_cooling_device *, unsigned long *);
  int (*get_cur_state)(struct thermal_cooling_device *, unsigned long *);
  int (*set_cur_state)(struct thermal_cooling_device *, unsigned long);
  int (*get_requested_power)(struct thermal_cooling_device *, u32 *);
  int (*state2power)(struct thermal_cooling_device *, unsigned long, u32 *);
  int (*power2state)(struct thermal_cooling_device *, u32, unsigned long *);
};
```
thermal_zone_device_register

- This function adds a new thermal zone device in the folder /sys/class/thermal.
- Bind all the thermal cooling devices registered at the same time.
- thermal_zone_device_unregister() must be called if the device is no longer needed.

- **type**: thermal zone type
- **trips**: the number of trip points the thermal zone support
- **mask**: trip points are writeable or not
- **devdata**: device data
- **ops**: standard thermal zone device callbacks
- **tzp**: thermal zone platform parameters
- **passive_delay**: delay to wait between polls when performing cooling
- **polling_delay**: delay to wait between polls when checking whether threshold points have been crossed or not
thermal_zone_of_sensor_register (device tree):

- This function will search the thermal zones in device tree and adds new sensor to DT thermal zone.

- **dev**: device node of the sensor
- **sensor_id**: sensor identifier
- **data**: a private pointer that will be passed back, when a temperature reading is needed.
- **ops**: struct thermal_zone_of_device_ops
Cooling device register function

- **thermal_cooling_device_register**
  - This function registers and creates *cooling_device* file in the folder `/sys/class/thermal/`
  - It checks and binds to the thermal zone.

- **type**: the cooling device name.
- **devdata**: device private data.
- **ops**: struct thermal_cooling_device_ops
Flow diagram of register thermal zone

1. System Boot
   - TMU Driver probe
     - Initialize device ops
       - get_temp = driver_func_1
       - set_emul_temp = driver_func_2
     - register_thermal_zone()
       - thermal_zone<->
         - user_read_temp_from_thermal_zone<->
           - display_temp_to_user
- Define `device_ops` and initialize “get_temp” pointer to the function which reads sensor temperature.
- Call function to register thermal zone in probe function of tmu driver.
- Pass `device_ops` which is already initialized.
- Driver might have `init_func` to initialize TMU.

- `<tmu_driver_probe>` happens multiple times.
- Each instance of TMU will be probed separately.
- Consumes additional memory and time to probe.
Demo of get temp and set emulation

```bash
# ls /sys/class/thermal/
thermal_zone0 thermal_zone1 thermal_zone2
thermal_zone3 thermal_zone4

# ls /sys/class/thermal/thermal_zone0/
  temp
  available_policies
    uevent
  emul_temp
  integral_cutoff
  k_d
  k_i
  k_po
  k Pu
  mode
  offset
  power
  slope
  subsystem

# cat /sys/class/thermal/thermal_zone0/temp
34000

# cat /sys/class/thermal/thermal_zone2/temp
34000
35000
36000
34000
34000

# cat /sys/class/thermal/thermal_zone2/temp
# cat /sys/class/thermal/thermal_zone2/temp
# cat /sys/class/thermal/thermal_zone2/temp
# cat /sys/class/thermal/thermal_zone2/temp
```

```bash
# cat /sys/class/thermal/thermal_zone0/
  temp
  available_policies
    uevent
  emul_temp
  integral_cutoff
  k_d
  k_i
  k_po
  k Pu
  mode
  offset
  power
  slope

# ls /sys/class/thermal/thermal_zone0/
  temp
  available_policies
    uevent
  emul_temp
  integral_cutoff
  k_d
  k_i
  k_po
  k Pu
  mode
  offset
  power
  slope

# ls /sys/class/thermal/thermal_zone0/
  temp
  available_policies
    uevent
  emul_temp
  integral_cutoff
  k_d
  k_i
  k_po
  k Pu
  mode
  offset
  power
  slope

# echo 1000 > /sys/class/thermal/thermal_zone0/emul_temp
# echo 1000 > /sys/class/thermal/thermal_zone1/emul_temp
# echo 1000 > /sys/class/thermal/thermal_zone2/emul_temp
# echo 1000 > /sys/class/thermal/thermal_zone2/emul_temp
# echo 1000 > /sys/class/thermal/thermal_zone2/emul_temp
# cat /sys/class/thermal/thermal_zone2/temp
1000
1000
1000
34000
34000
```
Policies:
- Thermal governor to manage over all thermal functionality

Power allocator:
- Closed loop control.
- Based on power budget, temperature and current power consumption.
- Implements PID controller with temperature as control input and power as control output.
- $k_d$, $k_i$, $k_po$, $k_pu$ constants for PID controller.
- integral_cutoff: cooling devices can't bring down temperature to the exact value the governor has requested. This field represents max offset allowed.

Step Wise:
- Open loop control.
- Based on temperature threshold and rate of change.
- Walk through each cooling state of each cooling device.
Demo of on trip points

- 0 ~ 7 threshold levels. Total 8 points
- Based on this threshold points, governor loop switch on the cooling devices and operate the cooling device in different state.
  - 0 ~ 6 are passive trip points.
  - 7th is critical trip point upon which power to the should be cut-off
- Trip temperature is displayed to user in milli-celsius range
- Hysteresis is the minimum change needed for thermal governed to take the next action

```bash
ls /sys/class/thermal/thermal_zone0/trip_point_
trip_point_0_hyst trip_point_2_hyst trip_point_4_hyst trip_point_6_hyst
trip_point_0_temp trip_point_2_temp trip_point_4_temp trip_point_6_temp
trip_point_0_type trip_point_2_type trip_point_4_type trip_point_6_type
trip_point_1_hyst trip_point_3_hyst trip_point_5_hyst trip_point_7_hyst
trip_point_1_temp trip_point_3_temp trip_point_5_temp trip_point_7_temp
trip_point_1_type trip_point_3_type trip_point_5_type trip_point_7_type
```

```bash
# cat /sys/class/thermal/thermal_zone0/trip_point_0_type
critical
```

```bash
# cat /sys/class/thermal/thermal_zone0/trip_point_0_temp
92000
95000
97000
100000
102000
105000
107000
112000
```

```bash
# cat /sys/class/thermal/thermal_zone0/trip_point_0_hyst
1000
1000
1000
1000
1000
1000
0
```
## Demo of on mode and type

```bash
# ls /sys/class/thermal/
thermal_zone0  thermal_zone1  thermal_zone2
thermal_zone3  thermal_zone4

# cat /sys/class/thermal/thermal_zone*/mode
enabled
enabled
enabled
enabled
disabled

cat /sys/class/thermal/thermal_zone*/type
cpu0-block-temp
cpu1-block-temp
cpu2-block-temp
cpu3-block-temp
gpu0-block-temp
```

**mode**
- Current mode of thermal zone
- Enabled : kernel thermal management is enabled
- Disabled: kernel thermal management doesn’t take action upon trip points. User space application take charge of thermal management

**type**
- Name of the thermal zone
- TMU in complex SoC may have Remote Sensor Interface additionally to Conventional TMU.
- Entire SoC can have a single TMU controller.
- If any block of SoC need to be monitored, then only Remote sensor has to be placed in that block.
- Sensor placed at different blocks of SoC is controlled and monitored by single TMU controller.
- Size and Cost can be reduced.
In Complex SoC, TMU Controller, Main Sensor and Remote sensors are represented as single unit.

Remote Sensors are connected to TMU controllers via Remote Sensor Interface.

Using Conventional TMU driver, can expose only Main Sensor to the user space.

User will not be able to read temperatures from Remote Sensors.

```c
/*Driver instance for TMU-0 */

static int <driver_probe_for_tmu_0>(struct platform_device *pdev)
{
    thermal_zone_device_register(<tmu_0>);
    tmu_init_func(<tmu_0>);
}

struct thermal_zone_device_ops <tmu_0> = {
    .get_temp = read_sensor_temperature_func();
};
```
- Define separate device_ops for each sensor and initialize "get_temp" to driver function to read temp.
- In driver probe, register thermal zone for each sensor and pass respective device_ops.
- `<tmu_driver_probe>` happens only once.
Current Thermal zone device register function

1. `thermal_zone_device_register`
2. set passive and polling delays
3. set trip parameters (type, temp, hyst)
4. binds cooling device
5. initialize thermal governor
6. creates thermal zone
7. function return
Any Questions?
THANK YOU