Reservoir Labs

Power API
2014.05.01

Sponsored by:
Defense Advanced Research Projects Agency
Microsystems Technology Office (MTO)
Program: Power Efficiency Revolution for Embedded Computing Technologies (PERFECT)
Issued by DARPA/CMO under Contract No: HR0011-12-C-0123

Contents

Chapter 1

Reservoir Labs Power API

1.1 Overview

The Power API is divided into high-level and low-level interfaces. The high-level interface allows a programmer or compiler to specify power and energy management goals and choose strategies to achieve these goals. The implementation of strategies and the means to track and enforce power management goals is the responsibility of the implementor of the API on a platform.

The low-level interface of the Power API is close to the hardware and provides direct control over DVFS settings for voltage islands.

The high-level goals of the API are as follows:

- Provide a cross-platform interface for compilers and programmers to control power and energy consumption
- Be concise, intuitive and make minimal assumptions about the underlying hardware
- Maintain a level of abstraction high enough to allow implementation in terms of DARPA PERFECT team APIs.
- Take advantage of features provided by leading edge task-based runtime environments

The API assumes that any system components bound to the same voltage and frequency settings are grouped together in an island. An island is the atomic unit for which frequency and voltage can be modified through the Power API.

1.2 High-Level Interface

The high-level interface of the Power API is accessed through an initialization function and the data structures passed to this function.

The programmer (or compiler) must set up three things at Power API initialization:

- A model of hardware behavior
- A speed adjustment policy
- A scheduling policy

See also

pwr_initialize()

Once configured, the combination of these 3 elements guides power and energy management decisions at program execution time. The 3 elements and associated data structures are described in the following sections. Power API implementations must define a default for each element on the targeted architecture.

1.2.1 Hardware Behavior

This defines the valid combinations of voltage and speed / speed level, possibly as functions of external factors such as the current temperature. Hardware behavior may be changed by hardware, software, or a combination of both.

See also

hw_behavior_t
pwr_hw_behavior()
pwr_change_hw_behavior()

Todo Clarify hardware behavior relationship with temperature / external factors.

Clarify definition of task, processing element.

Consider a near-threshold voltage architecture that may trade accuracy for power savings when voltage drops near threshold. An application that is resillient to errors would use a hardware behavior that allowed to use all voltage / frequency combinations supported by the architecture. An application that demands accurate results would use a hardware behavior that limited available voltage-frequency combinations to those well above threshold voltage.

1.2.2 Speed Policy

This defines a blanket policy for determining the speed level at which voltage islands are set.

Rather than exposing a notion of frequency, we expose a notion of speed level. This decision addresses the following:

- Permissible frequencies at discrete values
- Heterogeneity of architectures. Two different chips may have the same frequency but they won't have the same speed level (i.e., they won't execute the same piece of code in the same amount of time).

In the Power API, we define frequency, speed level, and speed:

- Frequency: The clock rate of a voltage island, in KHz
- Speed: A real number that corresponds to the absolute performance of a voltage island
- Speed Level: An integer greater than or equal to -1 that identifies a legal combination of voltage and frequency for a voltage island

See also

Speed Policies
pwr_request_speed_level()
pwr_modify_speed_level()
pwr_current_speed_level()
speed_policy_t

1.2.3 Scheduling Policy

This defines a blanket policy for the run-time assignment of tasks to processing elements.

See also

Scheduling Policies
scheduling_policy_t

Todo More options for scheduling policy. Specifically, space and time mapping.

1.3 Low-Level Interface

This interface allows the user to modify the speed level and voltage of an island. Speed level and voltage are not necessarily independent, so modifying one may modify the other.

A user is expected to be interested in upping the speed level or lowering the voltage knowing that the corresponding power consumption and speed will be negatively affected. An advanced user or compiler familiar with both island and application characteristics could modify speed and voltage in the same direction and still achieve power or performance gains. The goal of the low-level interface is to enable these modifications.

1.3.1 Speed Level

1.3.2 Agility

Agility is defined as the best and worst case amount of time it takes to switch from one speed level to another on a voltage island.

See also

pwr_agility()

1.4 Example Code

```
#include <power_api.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
    retval_t retval;
    long num_islands;
    island_id_t islands;

    // Initialize with default values
    retval = pwr_initialize(NULL, NULL, NULL);

    // Get island ids
    retval = pwr_num_islands(&num_islands);
    islands = (island_id_t*)malloc(num_islands*sizeof(island_id_t));
    retval = pwr_islands(islands);

    // Set each island to max speed
    for (long i = 0; i < num_islands; ++i) {
        retval = pwr_request_speed_level(islands[i], PWR_MAX_SPEED_LEVEL);
    }

    // Finalize
    retval = pwr_finalize();
    return 0;
}
```

Chapter 2

Todo List

Global pwr_agility (const island_id_t island, const speed_t from_level, const speed_t to_level, agility_t *best_case, agility_t *worst_case)

Decide on unit for agility. Currently ns because of cpufreq default. If agility in cycles, at what speed level? Full speed? `to_level`?

Global pwr_modify_speed_level (const island_id_t island, const int delta, const speed_level_t bottom)

Clarify behavior when an illegal speed level is requested

Clarify semantics of speed levels when multiple hardware behaviors are used

Global pwr_modify_voltage (const island_id_t island, const int delta)

Should we have a corresponding function to directly set voltage as we do with speed level?

Global pwr_num_speed_levels (const island_id_t island, long *num_speed_levels)

Map speed level to integer between 0 and 100 inclusive?

page Reservoir Labs Power API

Clarify hardware behavior relationship with temperature / external factors.

Clarify definition of task, processing element.

More options for scheduling policy. Specifically, space and time mapping.

Chapter 3

Module Documentation

3.1 Limitations

Defines

- #define PWR_MAX_ISLANDS (1024L∗1024L∗1024L)

  The maximum number of voltage islands in a system supported by the Power API.
- #define PWR_MAX_SPEED_LEVELS (1024L∗1024L)

  The maximum number of speed levels supported by the Power API.

3.2 Speed Levels

Defines

- #define PWR_POWER_GATE (-1)

  Power to voltage island is off.
- #define PWR_CLOCK_GATE (0)

  Clock signal to voltage island is off.
- #define PWR_MIN_SPEED_LEVEL (1)

  Voltage island minimum speed level.
- #define PWR_MAX_SPEED_LEVEL (LONG_MAX)

  Voltage island maximum speed level.

3.3 Speed Policies

Defines

- #define PWR_SPEED_FIXED_MIN (1)

  Pin all voltage islands to minimum speed level.
- #define PWR_SPEED_FIXED_MAX (2)

  Pin all voltage islands to maximum speed level.
- #define PWR_SPEED_AS_NEEDED (3)

  Dynamically adjust speed level of voltage island to meet demand.
- #define PWR_SPEED_LOWLEVEL_API (4)

  Dynamically adjust speed level with Power API calls only.

3.4 Scheduling Policies

Defines

- #define PWR_SCHED_DISTRIBUTE (1)

  Distribute tasks across processing elements such that hardware resource sharing between tasks is minimized.
- #define PWR_SCHED_CONCENTRATE (2)

  Distribute tasks across processing elements such that hardware resource sharing between tasks is maximized.
- #define PWR_SCHED_RANDOM (3)

  Distribute tasks across processing elements randomly.
- #define PWR_SCHED_EPS (4)

  Use energy proportional scheduling for space-time task mapping.

3.5 Unique Identifiers

Typedefs

- typedef long island_id_t

  Integral unique identifier of a voltage island.

3.6 Units

Typedefs

- typedef double voltage_t

  Voltage in volts.
- typedef long freq_t

  Frequency in KHz.
- typedef long speed_t

  Speed.
- typedef long speed_level_t

  Unit-less speed level.
- typedef long agility_t

  Agility in ns.
- typedef long power_t

  Power in watts.
- typedef long energy_t

  Energy in joules or microjoules (see function documentation)
- typedef long timestamp_t

  Time in seconds or nanoseconds (see function documentation)

3.7 High-Level Interface Data Structures

Data Structures

- struct hw_behavior_t

  Defines legal (island_id, voltage, frequency, speed, speed_level) tuples.

Typedefs

- typedef long speed_policy_t

  Defines policy for determining speed.
- typedef long scheduling_policy_t

  Defines policy for scheduling.

3.8 Error Codes

Defines

- #define PWR_ARCH_UNSUPPORTED (-3)

  Feature unsupported by hardware.
- #define PWR_UNIMPLEMENTED (-2)

  Feature not implemented.
- #define PWR_UNINITIALIZED (-1)

  Power API has not been initialized.
- #define PWR_OK (0)

  Command executed successfully.
- #define PWR_ERR (1)

  Unspecified error.
- #define PWR_UNAVAILABLE (2)

  Feature temporarily unavailable.
- #define PWR_REQUEST_DENIED (4)

  Request was denied.
- #define PWR_INIT_ERR (5)

  Unspecified error during initialization.
- #define PWR_FINAL_ERR (6)

  Unspecified error during finalization.
- #define PWR_ALREADY_INITIALIZED (7)

  Attempt to initialize API after it has been initialized.
- #define PWR_IO_ERR (8)

  Unspecified input / output error.
- #define PWR_UNSUPPORTED_SPEED_LEVEL (9)

  Speed level not supported by hardware or API.
- #define PWR_UNSUPPORTED_VOLTAGE (10)

  Voltage not supported by hardware or API.
- #define PWR_ALREADY_MINMAX (11)

  Feature is already set to minimum or maximum value.
- #define PWR_OVER_E_BUDGET (12)

  Request denied, over energy budget.
- #define PWR_OVER_P_BUDGET (13)

  Request denied, over power budget.
- #define PWR_OVER_T_BUDGET (14)

  Request denied, over thermal budget.
- #define PWR_INVALID_ISLAND (15)

  Specified island does not exist.

- #define PWR_DVFS_ERR (16)

  Unspecified error when changing voltage and/or frequency.
- #define PWR_OVERFLOW_ERR (17)

  Requested value overflow.

Typedefs

- typedef int retval_t

  Error codes returned by API functions.

3.9 High-Level Interface

Functions

- retval_t pwr_is_initialized (int *is_initialized)

  Checks if the Power API has been initialized.
- retval_t pwr_initialize (const hw_behavior_t *hw_behavior, const speed_policy_t *speed_policy, const scheduling_policy_t *scheduling_policy)

  Allocates resources used by the Power API.
- retval_t pwr_finalize ()

  Frees resources used by the Power API.
- retval_t pwr_ecount_finalize ()

  This method is a quick fix to a strange problem caused by re-initializing and finalizing the ecount environment multiple times.
- retval_t pwr_hw_behavior (hw_behavior_t *hw_behavior)

  Retrieve the currently active hardware behavior.
- retval_t pwr_change_hw_behavior (hw_behavior_t *hw_behavior)

  Change currently active hardware behavior.

3.9.1 Function Documentation

3.9.1.1 retval_t pwr_change_hw_behavior (hw_behavior_t * *hw_behavior*)

Parameters

hw_behavior	Pointer to new active hardware behavior struct

Return values

PWR_OK	Active hardware behavior successfully changed
PWR_UNINITIALIZED	Power API has not been initialized thus hardware behavior cannot be changed

3.9.1.2 retval_t pwr_ecount_finalize ( )

In particular, if the ec_finalize method is called in-between measurments then for some unknown reason measurments after ec_finalize always return zero.

3.9.1.3 retval_t pwr_finalize ( )

Must be called after all Power API functions have been called.

Return values

PWR_OK	All resources freed successfully
PWR_UNINITIALIZED	Power API has not been initialized thus cannot be finalized

3.9.1.4 retval_t pwr_hw_behavior (hw_behavior_t * *hw_behavior*)

Parameters

hw_behavior	Pointer to active hardware behavior struct

Return values

PWR_OK	Active hardware behavior returned successfully
PWR_UNINITIALIZED	Power API has not been initialized thus hardware behavior is unavailable

3.9.1.5 retval_t pwr_initialize (const hw_behavior_t * *hw_behavior*, const speed_policy_t * *speed_policy*, const scheduling_policy_t * *scheduling_policy*)

Must be called before any other function in the Power API can be used.

Parameters

hw_behavior	Relationship between voltage and speed on all voltage islands
speed_policy	Blanket policy for controlling voltage island speed levels
scheduling_policy	Blanket policy for runtime task scheduling

Return values

PWR_OK	Initialized with no errors
PWR_INIT_ERR	Initialization failed
PWR_ALREADY_INITIALIZED	API is initialized

3.9.1.6 retval_t pwr_is_initialized (int * *is_initialized*)

Parameters

is_initialized	`TRUE` if initialized, `FALSE` otherwise

Return values

PWR_OK	All cases

3.10 Low-Level Interface

Functions

- retval_t pwr_num_islands (long *num_islands)

  The number of voltage islands controlled by the Power API.
- retval_t pwr_islands (island_id_t *islands)

  Retrieve unique ID for all voltage islands.
- retval_t pwr_num_speed_levels (const island_id_t island, long *num_speed_levels)

  Number of discrete speed levels, not including power and clock gated states, supported by a voltage island.
- retval_t pwr_current_speed_level (const island_id_t island, speed_level_t *current_level)

  The current speed level of a voltage island.
- retval_t pwr_request_speed_level (const island_id_t island, const speed_level_t new_level)

  Requests speed level on the given voltage island.
- retval_t pwr_modify_speed_level (const island_id_t island, const int delta, const speed_level_t bottom)

  Requests a speed level modification on the given island.
- retval_t pwr_agility (const island_id_t island, const speed_t from_level, const speed_t to_level, agility_t *best_case, agility_t *worst_case)

  Calculates the cost of switching speed levels.
- retval_t pwr_modify_voltage (const island_id_t island, const int delta)

  Requests a voltage level modification of the given island.
- retval_t pwr_energy_counter (const island_id_t island, energy_t *e_j, energy_t *e_uj, timestamp_t *t_sec, timestamp_t *t_nsec)

  Value of monotonically increasing energy counter on the given island.

3.10.1 Function Documentation

3.10.1.1 retval_t pwr_agility (const island_id_t *island*, const speed_t *from_level*, const speed_t *to_level*, agility_t * *best_case*, agility_t * *worst_case*)

Todo Decide on unit for agility. Currently ns because of cpufreq default. If agility in cycles, at what speed level? Full speed? `to_level`?

Parameters

island	The island of interest
from_level	Starting speed level
to_level	Finishing speed level
best_case	Minimum cost in ns
worst_case	Maximum cost in ns

Return values

PWR_OK	Agility successfully returned
PWR_ERR	Agility not returned

3.10.1.2 retval_t pwr_current_speed_level (const island_id_t *island*, speed_level_t * *current_level*)

Parameters

island	The island to check speed level on
current_level	The current speed level of `island`

Return values

PWR_OK	Speed level has been returned
PWR_INVALID_ISLAND	The given ID does not correspond to a voltage island.
PWR_UNINITIALIZED	Power API has not been initialized

3.10.1.3 retval_t pwr_energy_counter (const island_id_t *island*, energy_t * *e_j*, energy_t * *e_uj*, timestamp_t * *t_sec*, timestamp_t * *t_nsec*)

Total energy consumed (J) since some undefined starting point is `e_j + e_uj*10-E6`. Elapsed time (sec) since some undefined starting point is `t_sec + t_-nsec*10E9`.

Parameters

island	The island to measure energy consumption on
e_j	Energy consumed since some unspecified starting point on `island`, joule component
e_uj	Energy consumed since some undefined starting point on `island`, microjoule component
t_sec	Time since some unspecified starting point, seconds component
t_nsec	Time since some unspecified starting point, nanoseconds component

Return values

PWR_OK	Energy counter successfuly returned
PWR_ERR	Energy counter not returned
PWR_OVERFLOW_ERR	Energy counter overflowed, inaccurate results provided

3.10.1.4 retval_t pwr_islands (island_id_t * *islands*)

Requires that `islands` points to at least `num_islands*sizeof(island_id_t)` bytes of memory.

Parameters

islands	An array of `island_id_t`

Return values

PWR_OK	Voltage island IDs have been returned
PWR_UNINITIALIZED	Power API has not been initialized

3.10.1.5 retval_t pwr_modify_speed_level (const island_id_t *island*, const int *delta*, const speed_level_t *bottom*)

`delta` can be positive or negative.

Parameters

island	The island to speed up or slow down
delta	The number of speed levels to increment by
bottom	The minimum legal speed level, can be one of PWR_POWER_GATE, PWR_CLOCK_GATE or PWR_MIN_SPEED_LEVEL

Todo Clarify behavior when an illegal speed level is requested

Clarify semantics of speed levels when multiple hardware behaviors are used

Returns

PWR_OK Speed level was successfully modified

Return values

PWR_UNSUPPORTED_SPEED_LEVEL	The requested speed level is not supported
PWR_ALREADY_MINMAX	Voltage island already at requested minimum or maximum speed level
PWR_DVFS_ERR	DVFS failed
PWR_INVALID_ISLAND	The given ID does not correspond to a voltage island.
PWR_OVER_E_BUDGET	Request denied, over energy budget (reserved for future use)
PWR_OVER_P_BUDGET	Request denied, over power budget (reserved for future use)
PWR_OVER_T_BUDGET	Request denied, over thermal budget (reserved for future use)
PWR_UNINITIALIZED	Power API has not been initialized

3.10.1.6 retval_t pwr_modify_voltage (const island_id_t *island*, const int *delta*)

`delta` can be positive or negative.

Todo Should we have a corresponding function to directly set voltage as we do with speed level?

Parameters

island	The island to change voltage on
delta	The number of voltage levels to modify by

Return values

PWR_OK	Voltage level successfully modified
PWR_ARCH_UNSUPPORTED	Voltage level cannot be modified
PWR_UNSUPPORTED_SPEED_LEVEL	The requested speed level is not supported
PWR_ALREADY_MINMAX	Voltage island already at requested minimum or maximum voltage
PWR_DVFS_ERR	DVFS failed
PWR_INVALID_ISLAND	The given ID does not correspond to a voltage island.
PWR_OVER_E_BUDGET	Request denied, over energy budget (reserved for future use)

PWR_OVER_P_BUDGET	Request denied, over power budget (reserved for future use)
PWR_OVER_T_BUDGET	Request denied, over thermal budget (reserved for future use)
PWR_UNINITIALIZED	Power API has not been initialized
PWR_ERR	Voltage level not successfully modified

3.10.1.7 retval_t pwr_num_islands (long * *num_islands*)

Parameters

num_islands	Total voltage islands controlled by the Power API

Return values

PWR_OK	Number of voltage islands has been returned
PWR_UNINITIALIZED	Power API has not been initialized

3.10.1.8 retval_t pwr_num_speed_levels (const island_id_t *island,* long * *num_speed_levels*)

The slowest speed level is '`1`' and speed levels increase monotonically until the fastest speed level at '`num_speed_levels`'.

Todo Map speed level to integer between 0 and 100 inclusive?

Parameters

island	The ID of the island to get speed level count for
num_speed_levels	The number of speed levels supported by `island`

Return values

PWR_OK	Number of speed levels has been returned
PWR_INVALID_ISLAND	The given ID does not correspond to a voltage island.
PWR_UNINITIALIZED	Power API has not been initialized

3.10.1.9 retval_t pwr_request_speed_level (const island_id_t *island*, const speed_level_t *new_level*)

Requires `-1 <= new_level <= num_speed_levels`. 'new_level == -1' requests power gating. 'new_level == 0' requests clock gating.

Parameters

island	The island to request speed level on
new_level	The requested speed level

Return values

PWR_OK	Speed level has been changed
PWR_UNSUPPORTED_SPEED_LEVEL	The requested speed level is not supported
PWR_ALREADY_MINMAX	Voltage island already at requested minimum or maximum speed level
PWR_DVFS_ERR	DVFS failed
PWR_INVALID_ISLAND	The given ID does not correspond to a voltage island.
PWR_OVER_E_BUDGET	Request denied, over energy budget (reserved for future use)
PWR_OVER_P_BUDGET	Request denied, over power budget (reserved for future use)
PWR_OVER_T_BUDGET	Request denied, over thermal budget (reserved for future use)
PWR_UNINITIALIZED	Power API has not been initialized

Chapter 4

Data Structure Documentation

4.1 e_data Struct Reference

Definition of struct to hold energy consumption and timing information.

`#include <ecount.h>`

Data Fields

- double **start_time_ns**
- double **stop_time_ns**
- long long **values** [NUM_EVENTS]
- char **names** [NUM_EVENTS][EC_MAX_STR_LEN]
- char **units** [NUM_EVENTS][EC_MIN_STR_LEN]

4.1.1 Detailed Description

Definition at line 39 of file ecount.h.

4.2 hw_behavior_t Struct Reference

Defines legal (island_id, voltage, frequency, speed, speed_level) tuples.

`#include <power_api.h>`

Data Fields

- long num_tuples

  Total number of (island_id, voltage, frequency, speed, speed_level) tuples.
- island_id_t * island

  Island that this tuple applies to (required)
- voltage_t * volts

  Voltage value of the tuple (required)
- freq_t * freq

  Frequency value of the tuple (required)
- speed_t * speed

  Speed value of the tuple (optional)
- speed_level_t * speed_level

  Speed level value of the tuple (required)

4.2.1 Detailed Description

Definition at line 332 of file power_api.h.

Chapter 5

File Documentation

5.1 power_api.h File Reference


```
#include <limits.h>
```

Data Structures

- struct hw_behavior_t

  Defines legal (island_id, voltage, frequency, speed, speed_level) tuples.

Defines

- #define PWR_MAX_ISLANDS (1024L∗1024L∗1024L)

  The maximum number of voltage islands in a system supported by the Power API.
- #define PWR_MAX_SPEED_LEVELS (1024L∗1024L)

  The maximum number of speed levels supported by the Power API.
- #define PWR_POWER_GATE (-1)

  Power to voltage island is off.
- #define PWR_CLOCK_GATE (0)

  Clock signal to voltage island is off.
- #define PWR_MIN_SPEED_LEVEL (1)

  Voltage island minimum speed level.
- #define PWR_MAX_SPEED_LEVEL (LONG_MAX)

  Voltage island maximum speed level.

- #define PWR_SPEED_FIXED_MIN (1)

  Pin all voltage islands to minimum speed level.
- #define PWR_SPEED_FIXED_MAX (2)

  Pin all voltage islands to maximum speed level.
- #define PWR_SPEED_AS_NEEDED (3)

  Dynamically adjust speed level of voltage island to meet demand.
- #define PWR_SPEED_LOWLEVEL_API (4)

  Dynamically adjust speed level with Power API calls only.
- #define PWR_SCHED_DISTRIBUTE (1)

  Distribute tasks across processing elements such that hardware resource sharing between tasks is minimized.
- #define PWR_SCHED_CONCENTRATE (2)

  Distribute tasks across processing elements such that hardware resource sharing between tasks is maximized.
- #define PWR_SCHED_RANDOM (3)

  Distribute tasks across processing elements randomly.
- #define PWR_SCHED_EPS (4)

  Use energy proportional scheduling for space-time task mapping.
- #define PWR_ARCH_UNSUPPORTED (-3)

  Feature unsupported by hardware.
- #define PWR_UNIMPLEMENTED (-2)

  Feature not implemented.
- #define PWR_UNINITIALIZED (-1)

  Power API has not been initialized.
- #define PWR_OK (0)

  Command executed successfully.
- #define PWR_ERR (1)

  Unspecified error.
- #define PWR_UNAVAILABLE (2)

  Feature temporarily unavailable.
- #define PWR_REQUEST_DENIED (4)

  Request was denied.
- #define PWR_INIT_ERR (5)

  Unspecified error during initialization.
- #define PWR_FINAL_ERR (6)

  Unspecified error during finalization.
- #define PWR_ALREADY_INITIALIZED (7)

  Attempt to initialize API after it has been initialized.
- #define PWR_IO_ERR (8)

  Unspecified input / output error.

- #define PWR_UNSUPPORTED_SPEED_LEVEL (9)

  Speed level not supported by hardware or API.
- #define PWR_UNSUPPORTED_VOLTAGE (10)

  Voltage not supported by hardware or API.
- #define PWR_ALREADY_MINMAX (11)

  Feature is already set to minimum or maximum value.
- #define PWR_OVER_E_BUDGET (12)

  Request denied, over energy budget.
- #define PWR_OVER_P_BUDGET (13)

  Request denied, over power budget.
- #define PWR_OVER_T_BUDGET (14)

  Request denied, over thermal budget.
- #define PWR_INVALID_ISLAND (15)

  Specified island does not exist.
- #define PWR_DVFS_ERR (16)

  Unspecified error when changing voltage and/or frequency.
- #define PWR_OVERFLOW_ERR (17)

  Requested value overflow.

Typedefs

- typedef long island_id_t

  Integral unique identifier of a voltage island.
- typedef double voltage_t

  Voltage in volts.
- typedef long freq_t

  Frequency in KHz.
- typedef long speed_t

  Speed.
- typedef long speed_level_t

  Unit-less speed level.
- typedef long agility_t

  Agility in ns.
- typedef long power_t

  Power in watts.
- typedef long energy_t

  Energy in joules or microjoules (see function documentation)
- typedef long timestamp_t

  Time in seconds or nanoseconds (see function documentation)

- typedef long speed_policy_t

  Defines policy for determining speed.
- typedef long scheduling_policy_t

  Defines policy for scheduling.
- typedef int retval_t

  Error codes returned by API functions.

Functions

- retval_t pwr_is_initialized (int *is_initialized)

  Checks if the Power API has been initialized.
- retval_t pwr_initialize (const hw_behavior_t *hw_behavior, const speed_policy_t *speed_policy, const scheduling_policy_t *scheduling_policy)

  Allocates resources used by the Power API.
- retval_t pwr_finalize ()

  Frees resources used by the Power API.
- retval_t pwr_ecount_finalize ()

  This method is a quick fix to a strange problem caused by re-initializing and finalizing the ecount environment multiple times.
- retval_t pwr_hw_behavior (hw_behavior_t *hw_behavior)

  Retrieve the currently active hardware behavior.
- retval_t pwr_change_hw_behavior (hw_behavior_t *hw_behavior)

  Change currently active hardware behavior.
- retval_t pwr_num_islands (long *num_islands)

  The number of voltage islands controlled by the Power API.
- retval_t pwr_islands (island_id_t *islands)

  Retrieve unique ID for all voltage islands.
- retval_t pwr_num_speed_levels (const island_id_t island, long *num_speed_levels)

  Number of discrete speed levels, not including power and clock gated states, supported by a voltage island.
- retval_t pwr_current_speed_level (const island_id_t island, speed_level_t *current_level)

  The current speed level of a voltage island.
- retval_t pwr_request_speed_level (const island_id_t island, const speed_level_t new_level)

  Requests speed level on the given voltage island.
- retval_t pwr_modify_speed_level (const island_id_t island, const int delta, const speed_level_t bottom)

  Requests a speed level modification on the given island.

- retval_t pwr_agility (const island_id_t island, const speed_t from_level, const speed_t to_level, agility_t *best_case, agility_t *worst_case)

  Calculates the cost of switching speed levels.
- retval_t pwr_modify_voltage (const island_id_t island, const int delta)

  Requests a voltage level modification of the given island.
- retval_t pwr_energy_counter (const island_id_t island, energy_t *e_j, energy_t *e_uj, timestamp_t *t_sec, timestamp_t *t_nsec)

  Value of monotonically increasing energy counter on the given island.

5.1.1 Detailed Description

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. Interface of the Power API for compiler-assisted energy-proportional scheduling.

Definition in file power_api.h.

5.2 power_api.h

```
00001 /**
00002   * Copyright 2013-14 Reservoir Labs, Inc.
00003   *
00004   * Licensed under the Apache License, Version 2.0 (the "License");
00005   * you may not use this file except in compliance with the License.
00006   * You may obtain a copy of the License at
00007   *
00008   *     http://www.apache.org/licenses/LICENSE-2.0
00009   *
00010   * Unless required by applicable law or agreed to in writing, software
00011   * distributed under the License is distributed on an "AS IS" BASIS,
00012   * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00013   * See the License for the specific language governing permissions and
00014   * limitations under the License.
00015   */
00016
00017
00018
00019 /**
00020   * @file power_api.h Interface of the Power API for compiler-assisted
00021   * energy-proportional scheduling.
00022   *
00023   * \mainpage Reservoir Labs Power API
00024   *
00025   * \section over Overview
00026   * The Power API is divided into high-level and low-level interfaces. The
00027   * high-level interface allows a programmer or compiler to specify power and
```

```
00028   * energy management goals and choose strategies to achieve
00029   * these goals. The implementation of strategies and the means to track and
00030   * enforce power management goals is the responsibility of the implementor of
00031   * the API on a platform.
00032   *
00033   * The low-level interface of the Power API is close to the hardware and
00034   * provides direct control over DVFS settings for voltage islands.
00035   *
00036   * The high-level goals of the API are as follows:
00037   *   - Provide a cross-platform interface for compilers and programmers to
00038   *     control power and energy consumption
00039   *   - Be concise, intuitive and make minimal assumptions about the
00040   *     underlying hardware
00041   *   - Maintain a level of abstraction high enough to allow implementation
00042   *     in terms of DARPA PERFECT team APIs.
00043   *   - Take advantage of features provided by leading edge task-based runtime
00044   *     environments
00045   *
00046   * The API assumes that any system components bound to the same voltage and
00047   * frequency settings are grouped together in an island. An island is
00048   * the atomic unit for which frequency and voltage can be modified through
00049   * the Power API.
00050   *
00051   *
00052   * \section hl_desc High-Level Interface
00053   *
00054   * The high-level interface of the Power API is accessed through an
00055   * initialization function and the data structures passed to this function.
00056   *
00057   * The programmer (or compiler) must set up three things at Power API
00058   * initialization:
00059
00060   *   - A model of hardware behavior
00061   *   - A speed adjustment policy
00062   *   - A scheduling policy
00063   *
00064   * @see pwr_initialize()
00065   *
00066   * Once configured, the combination of these 3 elements guides power and
00067   * energy management decisions at program execution time. The 3 elements
00068   * and associated data structures are described in the following sections.
00069   * Power API implementations must define a default for each element on the
00070   * targeted architecture.
00071   *
00072   *
00073   *
00074   * \subsection hw_behav Hardware Behavior
00075   *
00076   * This defines the valid combinations of voltage and speed / speed level,
00077   * possibly as functions of external factors such as the current temperature.
00078   * Hardware behavior may be changed by hardware, software, or a combination of

00079   * both.
00080   *
00081   * @see hw_behavior_t
00082   * @see pwr_hw_behavior()
00083   * @see pwr_change_hw_behavior()
00084   *
00085   * @todo Clarify hardware behavior relationship with temperature / external
00086   *       factors.
00087   *
00088   * Consider a near-threshold voltage architecture that may trade accuracy for
       power
00089   * savings when voltage drops near threshold. An application that is
       resillient
00090   * to errors would use a hardware behavior that allowed to use all voltage /
00091   * frequency combinations supported by the architecture. An application that
00092   * demands accurate results would use a hardware behavior that limited
```

```
      available
00093  * voltage-frequency combinations to those well above threshold voltage.
00094  *
00095  *
00096  *
00097  * \subsection speed_policy Speed Policy
00098  *
00099  * This defines a blanket policy for determining the speed level at which
00100  * voltage islands are set.
00101  *
00102  * Rather than exposing a notion of frequency, we expose a notion of speed
00103  * level. This decision addresses the following:
00104  *  - Permissible frequencies at discrete values
00105  *  - Heterogeneity of architectures. Two different chips may have the same
00106  *    frequency but they won't have the same speed level (i.e., they won't
00107  *    execute the same piece of code in the same amount of time).
00108  *
00109  * In the Power API, we define frequency, speed level, and speed:
00110  *
00111  *  - Frequency: The clock rate of a voltage island, in KHz
00112  *  - Speed: A real number that corresponds to the absolute performance of a
00113  *           voltage island
00114  *  - Speed Level: An integer greater than or equal to -1 that identifies a
00115  *                 legal combination of voltage and frequency for a voltage
00116  *                 island
00117  *
00118  * @see speed_policies
00119  * @see pwr_request_speed_level()
00120  * @see pwr_modify_speed_level()
00121  * @see pwr_current_speed_level()
00122  * @see speed_policy_t
00123  *
00124  *
00125  *
00126  * \subsection scheduling Scheduling Policy
00127  *
00128  * This defines a blanket policy for the run-time assignment of tasks to
00129  * processing elements.
00130  *
00131  * @see scheduling_policies
00132  * @see scheduling_policy_t
00133  *
00134  * @todo More options for scheduling policy. Specifically, space and time
00135  *       mapping.
00136  * @todo Clarify definition of task, processing element.
00137  *
00138  *
00139  *
00140  * \section ll_desc Low-Level Interface
00141  *
00142  * This interface allows the user to modify the speed level and voltage of an
00143  * island. Speed level and voltage are not necessarily independent, so
00144  * modifying one may modify the other.
00145  *
00146  * A user is expected to be interested in upping the speed level or lowering
00147  * the voltage knowing that the corresponding power consumption and speed will

00148  * be negatively affected. An advanced user or compiler familiar with both
00149  * island and application characteristics could modify speed and voltage in
00150  * the same direction and still achieve power or performance gains. The goal
00151  * of the low-level interface is to enable these modifications.
00152  *
00153  *
00154  * \subsection speedlevel Speed Level
00155  *
00156  * \subsection agility Agility
00157  * Agility is defined as the best and worst case amount of time it takes to
00158  * switch from one speed level to another on a voltage island.
```

```
00159   *
00160   * @see pwr_agility()
00161   *
00162   *
00163   *
00164   *
00165   *
00166   * \section example Example Code
00167   *
00168   * \code
00169 #include <power_api.h>
00170 #include <stdio.h>
00171 #include <stdlib.h>
00172
00173 int main(int argc, char* argv[]) {
00174     retval_t retval;
00175     long num_islands;
00176     island_id_t islands;
00177
00178     // Initialize with default values
00179     retval = pwr_initialize(NULL, NULL, NULL);
00180
00181     // Get island ids
00182     retval = pwr_num_islands(&num_islands);
00183     islands = (island_id_t*)malloc(num_islands*sizeof(island_id_t));
00184     retval = pwr_islands(islands);
00185
00186     // Set each island to max speed
00187     for (long i = 0; i < num_islands; ++i) {
00188         retval = pwr_request_speed_level(islands[i], PWR_MAX_SPEED_LEVEL);
00189     }
00190
00191     // Finalize
00192     retval = pwr_finalize();
00193     return 0;
00194 }
00195 \endcode
00196   *
00197   *
00198   *
00199   *
00200   *
00201   *
00202   *
00203   */
00204 #ifndef POWERAPI_H
00205 #define POWERAPI_H
00206 #ifdef __cplusplus
00207 extern "C" {
00208 #endif
00209
00210 #include <limits.h>
00211
00212 //====------------------------------------------------------------------------
00213 // Constants
00214 //------------------------------------------------------------------------
00215 /** \addtogroup limits Limitations
00216   * @{
00217   */
00218
00219 /** The maximum number of voltage islands in a system supported by the Power
       API */
00220 #define PWR_MAX_ISLANDS (1024L*1024L*1024L)
00221
00222 /** The maximum number of speed levels supported by the Power API */
00223 #define PWR_MAX_SPEED_LEVELS (1024L*1024L)
00224 /** @} */
00225
```

```
00226
00227
00228 /** \addtogroup speed_control Speed Levels
00229   * @{
00230   */
00231 /** Power to voltage island is off */
00232 #define PWR_POWER_GATE (-1)
00233
00234 /** Clock signal to voltage island is off */
00235 #define PWR_CLOCK_GATE ( 0)
00236
00237 /** Voltage island minimum speed level */
00238 #define PWR_MIN_SPEED_LEVEL (1)
00239
00240 /** Voltage island maximum speed level */
00241 #define PWR_MAX_SPEED_LEVEL (LONG_MAX)
00242 /** @} */
00243
00244
00245
00246 /** \addtogroup speed_policies Speed Policies
00247   * @{
00248   */
00249 /** Pin all voltage islands to minimum speed level */
00250 #define PWR_SPEED_FIXED_MIN (1)
00251
00252 /** Pin all voltage islands to maximum speed level */
00253 #define PWR_SPEED_FIXED_MAX (2)
00254
00255 /** Dynamically adjust speed level of voltage island to meet demand */
00256 #define PWR_SPEED_AS_NEEDED (3)
00257
00258 /** Dynamically adjust speed level with Power API calls only */
00259 #define PWR_SPEED_LOWLEVEL_API (4)
00260 /** @} */
00261
00262
00263
00264 /** \addtogroup scheduling_policies Scheduling Policies
00265   * @{
00266   */
00267 /** Distribute tasks across processing elements such that hardware resource
00268     sharing between tasks is minimized */
00269 #define PWR_SCHED_DISTRIBUTE (1)
00270
00271 /** Distribute tasks across processing elements such that hardware resource
00272     sharing between tasks is maximized */
00273 #define PWR_SCHED_CONCENTRATE (2)
00274
00275 /** Distribute tasks across processing elements randomly */
00276 #define PWR_SCHED_RANDOM (3)
00277
00278 /** Use energy proportional scheduling for space-time task mapping */
00279 #define PWR_SCHED_EPS (4)
00280 /** @} */
00281
00282 //====------------------------------------------------------------------------
00283 // Typedefs
00284 //------------------------------------------------------------------------====
00285
00286 /** \addtogroup id Unique Identifiers
00287   * @{
00288   */
00289 /** Integral unique identifier of a voltage island */
00290 typedef long island_id_t;
00291
00292 /** @} */
00293
```

```
00294 /** \addtogroup units Units
00295   * @{
00296   */
00297 /** Voltage in volts */
00298 typedef double voltage_t;
00299
00300 /** Frequency in KHz */
00301 typedef long freq_t;
00302
00303 /** Speed */
00304 typedef long speed_t;
00305
00306 /** Unit-less speed level */
00307 typedef long speed_level_t;
00308
00309 /** Agility in ns */
00310 typedef long agility_t;
00311
00312 /** Power in watts */
00313 typedef long power_t;
00314
00315 /** Energy in joules or microjoules (see function documentation) */
00316 typedef long energy_t;
00317
00318 /** Time in seconds or nanoseconds (see function documentation) */
00319 typedef long timestamp_t;
00320 /** @} */
00321
00322 //====-------------------------------------------------------------------------
00323 // High-level interface data structures
00324 //----------------------------------------------------------------------------
00325 /** \addtogroup hli High-Level Interface Data Structures
00326   * @{
00327   */
00328
00329 /**
00330   * Defines legal (island_id, voltage, frequency, speed, speed_level) tuples
00331   */
00332 typedef struct hw_behavior {
00333     /** Total number of (island_id, voltage, frequency, speed, speed_level)
      tuples */
00334     long num_tuples;
00335     /** Island that this tuple applies to (required) */
00336     island_id_t* island;
00337     /** Voltage value of the tuple (required) */
00338     voltage_t* volts;
00339     /** Frequency value of the tuple (required) */
00340     freq_t* freq;
00341     /** Speed value of the tuple (optional) */
00342     speed_t* speed;
00343     /** Speed level value of the tuple (required) */
00344     speed_level_t* speed_level;
00345 } hw_behavior_t;
00346
00347 /**
00348   * Defines policy for determining speed.
00349   */
00350 typedef long speed_policy_t;
00351
00352
00353 /**
00354   * Defines policy for scheduling.
00355   */
00356 typedef long scheduling_policy_t;
00357
00358 /** @} */
00359
00360
```

```
00361 //====------------------------------------------------------------------------
00362 // Error codes
00363 //------------------------------------------------------------------------
00364
00365 /** \addtogroup errors Error Codes
00366   * @{
00367   */
00368
00369 /** Error codes returned by API functions */
00370 typedef int  retval_t;
00371
00372 /** Feature unsupported by hardware */
00373 #define PWR_ARCH_UNSUPPORTED (-3)
00374
00375 /** Feature not implemented */
00376 #define PWR_UNIMPLEMENTED (-2)
00377
00378 /** Power API has not been initialized */
00379 #define PWR_UNINITIALIZED (-1)
00380
00381 /** Command executed successfully  */
00382 #define PWR_OK (0)
00383
00384 /** Unspecified error */
00385 #define PWR_ERR (1)
00386
00387 /** Feature temporarily unavailable */
00388 #define PWR_UNAVAILABLE (2)
00389
00390 /** Request was denied */
00391 #define PWR_REQUEST_DENIED (4)
00392
00393 /** Unspecified error during initialization */
00394 #define PWR_INIT_ERR (5)
00395
00396 /** Unspecified error during finalization */
00397 #define PWR_FINAL_ERR (6)
00398
00399 /** Attempt to initialize API after it has been initialized */
00400 #define PWR_ALREADY_INITIALIZED (7)
00401
00402 /** Unspecified input / output error */
00403 #define PWR_IO_ERR (8)
00404
00405 /** Speed level not supported by hardware or API */
00406 #define PWR_UNSUPPORTED_SPEED_LEVEL (9)
00407
00408 /** Voltage not supported by hardware or API */
00409 #define PWR_UNSUPPORTED_VOLTAGE (10)
00410
00411 /** Feature is already set to minimum or maximum value */
00412 #define PWR_ALREADY_MINMAX (11)
00413
00414 /** Request denied, over energy budget */
00415 #define PWR_OVER_E_BUDGET (12) // reserved for future
00416
00417 /** Request denied, over power budget */
00418 #define PWR_OVER_P_BUDGET (13) // reserved for future
00419
00420 /** Request denied, over thermal budget */
00421 #define PWR_OVER_T_BUDGET (14) // reserved for future
00422
00423 /** Specified island does not exist */
00424 #define PWR_INVALID_ISLAND (15)
00425
00426 /** Unspecified error when changing voltage and/or frequency */
00427 #define PWR_DVFS_ERR (16)
00428
```

```
00429 /** Requested value overflow */
00430 #define PWR_OVERFLOW_ERR (17)
00431 /** @} */
00432
00433 //====------------------------------------------------------------------------
00434 // High-Level Interface
00435 //------------------------------------------------------------------------
00436
00437 /** \addtogroup highlevel High-Level Interface
00438   * @{
00439   */
00440
00441 /**
00442   * Checks if the Power API has been initialized
00443   *
00444   * @param is_initialized  <code>TRUE</code> if initialized, <code>FALSE</code>
      otherwise
00445   *
00446   * @retval #PWR_OK  All cases
00447   */
00448 retval_t pwr_is_initialized(int* is_initialized);
00449
00450
00451 /**
00452   * Allocates resources used by the Power API
00453   *
00454   * Must be called before any other function in the Power API can be used.
00455   *
00456   * @param hw_behavior Relationship between voltage and speed on all voltage
00457                       islands
00458   * @param speed_policy  Blanket policy for controlling voltage island speed
00459                         levels
00460   * @param scheduling_policy Blanket policy for runtime task scheduling
00461   *
00462   * @retval #PWR_OK  Initialized with no errors
00463   * @retval #PWR_INIT_ERR  Initialization failed
00464   * @retval #PWR_ALREADY_INITIALIZED  API is initialized
00465   */
00466 retval_t pwr_initialize(const hw_behavior_t* hw_behavior,
00467                         const speed_policy_t* speed_policy,
00468                         const scheduling_policy_t* scheduling_policy);
00469
00470
00471 /**
00472   * Frees resources used by the Power API
00473   *
00474   * Must be called after all Power API functions have been called.
00475   *
00476   * @retval #PWR_OK  All resources freed successfully
00477   * @retval #PWR_UNINITIALIZED  Power API has not been initialized thus cannot
00478   *                             be finalized
00479   */
00480 retval_t pwr_finalize();
00481
00482 /**
00483   * This method is a quick fix to a strange problem caused by re-initializing
00484   * and finalizing the ecount environment multiple times. In particular,
00485   * if the ec_finalize method is called in-between measurments then for some
00486   * unknown reason measurments after ec_finalize always return zero.
00487   */
00488 retval_t pwr_ecount_finalize();
00489
00490 /**
00491   * Retrieve the currently active hardware behavior
00492   *
00493   * @param hw_behavior  Pointer to active hardware behavior struct
00494   *
00495   * @retval #PWR_OK  Active hardware behavior returned successfully
```

```
00496   * @retval #PWR_UNINITIALIZED  Power API has not been initialized thus
       hardware
00497   *                            behavior is unavailable
00498   */
00499 retval_t pwr_hw_behavior(hw_behavior_t* hw_behavior);
00500
00501 /**
00502   * Change currently active hardware behavior
00503   *
00504   * @param hw_behavior Pointer to new active hardware behavior struct
00505   *
00506   * @retval #PWR_OK Active hardware behavior successfully changed
00507   * @retval #PWR_UNINITIALIZED  Power API has not been initialized thus
       hardware
00508   *                             behavior cannot be changed
00509   */
00510 retval_t pwr_change_hw_behavior(hw_behavior_t* hw_behavior);
00511
00512
00513 /** @} */
00514
00515 //====--------------------------------------------------------------------------
00516 // Low-Level Interface
00517 //----------------------------------------------------------------------------
00518
00519 /** \addtogroup lowlevel Low-Level Interface
00520   * @{
00521   */
00522 /**
00523   * The number of voltage islands controlled by the Power API
00524   *
00525   * @param num_islands  Total voltage islands controlled by the Power API
00526   *
00527   * @retval #PWR_OK  Number of voltage islands has been returned
00528   * @retval #PWR_UNINITIALIZED  Power API has not been initialized
00529   */
00530 retval_t pwr_num_islands(long* num_islands);
00531
00532
00533 /**

00534   * Retrieve unique ID for all voltage islands
00535   *
00536   * Requires that <code>islands</code> points to at least
00537   * <code>num_islands*sizeof(island_id_t)</code> bytes of memory.
00538   *
00539   * @param islands An array of <code>island_id_t</code>
00540   *
00541   * @retval #PWR_OK  Voltage island IDs have been returned
00542   * @retval #PWR_UNINITIALIZED  Power API has not been initialized
00543   */

00544 retval_t pwr_islands(island_id_t* islands);
00545
00546
00547 /**
00548   * Number of discrete speed levels, not including power and clock gated
       states,
00549   * supported by a voltage island
00550   *
00551   * The slowest speed level is '<code>1</code>' and speed levels increase
00552   * monotonically until the fastest speed level at
00553   * '<code>num_speed_levels</code>'.
00554   *
00555   * @todo Map speed level to integer between 0 and 100 inclusive?
00556   *
00557   * @param island  The ID of the island to get speed level count for
00558   * @param num_speed_levels  The number of speed levels supported by
```

```
      <code>island</code>
00559   *
00560   * @retval #PWR_OK  Number of speed levels has been returned
00561   * @retval #PWR_INVALID_ISLAND  The given ID does not correspond to a voltage
      island.
00562   * @retval #PWR_UNINITIALIZED  Power API has not been initialized
00563   */
00564 retval_t pwr_num_speed_levels(const island_id_t island, long* num_speed_levels)
      ;
00565
00566
00567 /**
00568   * The current speed level of a voltage island
00569   *
00570   * @param island  The island to check speed level on
00571   * @param current_level  The current speed level of <code>island</code>
00572   *
00573   * @retval #PWR_OK  Speed level has been returned
00574   * @retval #PWR_INVALID_ISLAND  The given ID does not correspond to a voltage
      island.
00575   * @retval #PWR_UNINITIALIZED  Power API has not been initialized
00576   */
00577 retval_t pwr_current_speed_level(const island_id_t island, speed_level_t*
      current_level);
00578
00579
00580 /**
00581   * Requests speed level on the given voltage island
00582   *
00583   * Requires <code>-1 <= new_level <= num_speed_levels</code>.
00584   * '<code>new_level == -1</code>' requests power gating.
00585   * '<code>new_level == 0</code>' requests clock gating.
00586   *
00587   * @param island  The island to request speed level on
00588   * @param new_level  The requested speed level
00589   *
00590   * @retval #PWR_OK  Speed level has been changed
00591   * @retval #PWR_UNSUPPORTED_SPEED_LEVEL  The requested speed level is not
      supported
00592   * @retval #PWR_ALREADY_MINMAX  Voltage island already at requested minimum or
      maximum speed level
00593   * @retval #PWR_DVFS_ERR  DVFS failed
00594   * @retval #PWR_INVALID_ISLAND  The given ID does not correspond to a voltage
      island.
00595   * @retval #PWR_OVER_E_BUDGET Request denied, over energy budget (reserved for
      future use)
00596   * @retval #PWR_OVER_P_BUDGET Request denied, over power budget (reserved for
      future use)
00597   * @retval #PWR_OVER_T_BUDGET Request denied, over thermal budget (reserved
      for future use)
00598   * @retval #PWR_UNINITIALIZED  Power API has not been initialized
00599   */
00600 retval_t pwr_request_speed_level(const island_id_t island, const speed_level_t
      new_level);
00601
00602
00603 /**
00604   * Requests a speed level modification on the given island
00605   *
00606   * <code>delta</code> can be positive or negative.
00607   *
00608   * @param island  The island to speed up or slow down
00609   * @param delta  The number of speed levels to increment by
00610   * @param bottom  The minimum legal speed level, can be one of
      #PWR_POWER_GATE,
00611   *                #PWR_CLOCK_GATE or #PWR_MIN_SPEED_LEVEL
00612   *
00613   * @todo Clarify behavior when an illegal speed level is requested
```

```
00614   * @todo Clarify semantics of speed levels when multiple hardware behaviors
00615   *       are used
00616   *
00617   * @return #PWR_OK  Speed level was successfully modified
00618   * @retval #PWR_UNSUPPORTED_SPEED_LEVEL  The requested speed level is not
       supported
00619   * @retval #PWR_ALREADY_MINMAX  Voltage island already at requested minimum or
       maximum speed level
00620   * @retval #PWR_DVFS_ERR  DVFS failed
00621   * @retval #PWR_INVALID_ISLAND  The given ID does not correspond to a voltage
       island.
00622   * @retval #PWR_OVER_E_BUDGET Request denied, over energy budget (reserved for
       future use)
00623   * @retval #PWR_OVER_P_BUDGET Request denied, over power budget (reserved for
       future use)
00624   * @retval #PWR_OVER_T_BUDGET Request denied, over thermal budget (reserved
       for future use)
00625   * @retval #PWR_UNINITIALIZED  Power API has not been initialized
00626   */
00627 retval_t pwr_modify_speed_level(const island_id_t      island,
00628                                 const int              delta,
00629                                 const speed_level_t    bottom);
00630
00631 /**
00632   * Calculates the cost of switching speed levels
00633   *
00634   * @todo Decide on unit for agility. Currently ns because of cpufreq default.
00635   *       If agility in cycles, at what speed level? Full speed?
00636   *       <code>to_level</code>?
00637   *
00638   * @param island  The island of interest
00639   * @param from_level  Starting speed level
00640   * @param to_level  Finishing speed level
00641   * @param best_case  Minimum cost in ns
00642   * @param worst_case Maximum cost in ns
00643   *
00644   * @retval #PWR_OK Agility successfully returned
00645   * @retval #PWR_ERR Agility not returned
00646   */
00647 retval_t pwr_agility(const island_id_t      island,
00648                      const speed_t          from_level,
00649                      const speed_t          to_level,
00650                            agility_t*       best_case,
00651                            agility_t*       worst_case);
00652
00653 /**
00654   * Requests a voltage level modification of the given island
00655   *
00656   * <code>delta</code> can be positive or negative.
00657   *
00658   * @todo Should we have a corresponding function to directly set voltage as
00659   *       we do with speed level?
00660   *
00661   * @param island  The island to change voltage on
00662   * @param delta  The number of voltage levels to modify by
00663   *
00664   * @retval #PWR_OK  Voltage level successfully modified
00665   * @retval #PWR_ARCH_UNSUPPORTED  Voltage level cannot be modified
00666   * @retval #PWR_UNSUPPORTED_SPEED_LEVEL  The requested speed level is not
       supported
00667   * @retval #PWR_ALREADY_MINMAX  Voltage island already at requested minimum or
       maximum voltage
00668   * @retval #PWR_DVFS_ERR  DVFS failed
00669   * @retval #PWR_INVALID_ISLAND  The given ID does not correspond to a voltage
       island.
00670   * @retval #PWR_OVER_E_BUDGET Request denied, over energy budget (reserved for
       future use)
00671   * @retval #PWR_OVER_P_BUDGET Request denied, over power budget (reserved for
```

```
      future use)
00672   * @retval #PWR_OVER_T_BUDGET Request denied, over thermal budget (reserved
      for future use)
00673   * @retval #PWR_UNINITIALIZED  Power API has not been initialized
00674   * @retval #PWR_ERR Voltage level not successfully modified
00675   */
00676 retval_t pwr_modify_voltage(const island_id_t island, const int delta);
00677
00678 /**
00679   * Value of monotonically increasing energy counter on the given island.
00680   * Total energy consumed (J) since some undefined starting point is
00681   * <code>e_j + e_uj*10E6</code>. Elapsed time (sec) since some undefined
00682   * starting point is <code>t_sec + t_nsec*10E9</code>.
00683   *
00684   * @param island  The island to measure energy consumption on
00685   * @param e_j  Energy consumed since some unspecified starting point on
00686                 <code>island</code>, joule component
00687   * @param e_uj  Energy consumed since some undefined starting point on
00688                  <code>island</code>, microjoule component
00689   * @param t_sec  Time since some unspecified starting point, seconds component
00690   * @param t_nsec  Time since some unspecified starting point, nanoseconds
      component
00691   *
00692   * @retval #PWR_OK  Energy counter successfuly returned
00693   * @retval #PWR_ERR  Energy counter not returned
00694   * @retval #PWR_OVERFLOW_ERR  Energy counter overflowed, inaccurate results
      provided
00695   */
00696 retval_t pwr_energy_counter(const island_id_t      island,
00697                                 energy_t*        e_j,
00698                                 energy_t*        e_uj,
00699                                 timestamp_t*     t_sec,
00700                                 timestamp_t*     t_nsec);
00701 /** @} */
00702 #ifdef __cplusplus
00703 }
00704 #endif
00705 #endif
```