AMDPowerProfile API User Guide

Release v1.2

AMD Developer Tools Team

May 02, 2018

CONTENTS

CHAPTER

ONE

AMDPOWERPROFILEAPI LIBRARY

1.1 Introduction

AMDPowerProfileApi library provides APIs to configure, collect and report the supported power profiling counters on various AMD platforms. The AMDPowerProfile API library is useful to analyze the energy efficiency of systems based on AMD CPUs, APUs and dGPUs (Discrete GPU). These APIs provide interface to read the power, thermal and frequency characteristics of APU/dGPU and their subcomponents. These APIs are targeted for software developers who want to write their own application to sample the power counters based on their specific use case.

1.2 APIs

1.2.1 AMDTPwrProfileInitialize

This API loads and initializes the AMDT Power Profile drivers. This API should be the first one to be called.

AMDTResult **`AMDTPwrProfileInitialize`**(*AMDTPwrProfileMode profileMode*)

Parameters

- `profileMode`: Client should select any one of the predefined profile modes that are defined in *`AMDTPwrProfileMode`*.

Returns

- `AMDT_STATUS_OK`: Success
- `AMDT_ERROR_INVALIDARG`: An invalid profileMode parameter was passed
- `AMDT_ERROR_DRIVER_UNAVAILABLE`: Driver not available
- `AMDT_DRIVER_VERSION_MISMATCH`: Mismatch between the expected and installed driver versions
- `AMDT_ERROR_PLATFORM_NOT_SUPPORTED`: Platform not supported
- `AMDT_WARN_SMU_DISABLED`: SMU is disabled and hence power and thermal values provided by SMU will not be available
- `AMDT_WARN_IGPU_DISABLED`: Internal GPU is disabled
- `AMDT_ERROR_FAIL`: An internal error occurred
- `AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED`: Previous session was not closed.

1.2.2 AMDTPwrGetSupportedCounters

This API provides the list of counters supported by the platform. The pointers returned will be valid till the client calls `AMDTPwrProfileClose()` function.

AMDTResult **`AMDTPwrGetSupportedCounters`**(AMDTUInt32 **pNumCounters*, *AMDTPwrCounterDesc **ppCounterDescs*)

Parameters

- `pNumCounters`: Number of counters supported by the device
- `ppCounterDescs`: Description of each counter supported by the device

Returns

- `AMDT_STATUS_OK`: On Success
- `AMDT_ERROR_INVALIDARG`: NULL pointer was passed as ppCounterDescs or pNumCounters parameters
- `AMDT_ERROR_DRIVER_UNINITIALIZED`: `AMDTPwrProfileInitialize()` function was neither called nor successful
- `AMDT_ERROR_INVALID_DEVICEID`: invalid deviceId parameter was passed
- `AMDT_ERROR_OUTOFMEMORY`: Failed to allocate required memory
- `AMDT_ERROR_FAIL`: An internal error occurred

1.2.3 AMDTPwrGetCounterId

This API provides the counter id for a basic counter.

AMDTResult **`AMDTPwrGetCounterId`**(*AMDTCounter counter*, AMDTUInt32 **pCounterId*)

Parameters

- `counter`: supported counter to get the counter id
- `pCounterId`: counter id of counter.

Returns

- `AMDT_STATUS_OK`: On Success
- `AMDT_ERROR_INVALIDARG`: NULL pointer was passed as pCounterDesc parameter
- `AMDT_ERROR_DRIVER_UNINITIALIZED`: `AMDTPwrProfileInitialize()` function was neither called nor successful
- `AMDT_ERROR_INVALID_COUNTERID`: Invalid counterId parameter was passed
- `AMDT_ERROR_FAIL`: An internal error occurred

1.2.4 AMDTPwrGetCounterDesc

This API provides the description for the given counter index.

AMDTResult **`AMDTPwrGetCounterDesc`**(AMDTUInt32 *counterId*, *AMDTPwrCounterDesc *pCounterDesc*)

Parameters

- `counterId`: Counter index
- `pCounterDesc`: Description of the counter which index is counterId

Returns

- `AMDT_STATUS_OK`: On Success
- `AMDT_ERROR_INVALIDARG`: NULL pointer was passed as pCounterDesc parameter
- `AMDT_ERROR_DRIVER_UNINITIALIZED`: *`AMDTPwrProfileInitialize()`* function was neither called nor successful
- `AMDT_ERROR_INVALID_COUNTERID`: Invalid counterId parameter was passed
- `AMDT_ERROR_FAIL`: An internal error occurred

1.2.5 AMDTPwrEnableCounter

This API will enable the counter to be sampled. This API cannot be used once profile is started.

AMDTResult **AMDTPwrEnableCounter**(AMDTUInt32 *counterId*)

Parameters

- `counterId`: Counter index

Returns

- `AMDT_STATUS_OK`: On Success
- `AMDT_ERROR_DRIVER_UNINITIALIZED`: *`AMDTPwrProfileInitialize()`* function was neither called nor successful
- `AMDT_ERROR_INVALID_COUNTERID`: Invalid counterId parameter was passed
- `AMDT_ERROR_COUNTER_ALREADY_ENABLED`: Specified counter is already enabled
- `AMDT_ERROR_PROFILE_ALREADY_STARTED`: Counters cannot be enabled on the fly when the profile is already started
- `AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED`: Previous session was not closed
- `AMDT_ERROR_COUNTER_NOT_ACCESSIBLE`: Counter is not accessible
- `AMDT_ERROR_FAIL`: An internal error occurred

1.2.6 AMDTPwrSetTimerSamplingPeriod

This API will set the driver to periodically sample the counter values and store them in a buffer. This cannot be called once the profile run is started. This API is not required to call if AMDTPwrProfileInitialize API is called with AMDT_PWR_MODE_INSTANT_COUNTER as profileMode.

AMDTResult **AMDTPwrSetTimerSamplingPeriod**(AMDTUInt32 *interval*)

Parameters

- `interval`: sampling period in millisecond

Returns

- `AMDT_STATUS_OK`: On Success

- `AMDT_ERROR_INVALIDARG`: Invalid interval value was passed
- `AMDT_ERROR_DRIVER_UNINITIALIZED`: *`AMDTPwrProfileInitialize()`* function was neither called nor successful
- `AMDT_ERROR_PROFILE_ALREADY_STARTED`: Timer interval cannot be changed when the profile is already started
- `AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED`: Previous session was not closed
- `AMDT_ERROR_FAIL`: An internal error occurred

1.2.7 AMDTPwrStartProfiling

This API will start the profiling and the driver will collect the data at regular interval specified by *`AMDTPwrSetTimerSamplingPeriod()`*. This has to be called after enabling the required counters by using *`AMDTPwrEnableCounter()`*.

AMDTResult **`AMDTPwrStartProfiling`**()

Returns

- `AMDT_STATUS_OK`: On Success
- `AMDT_ERROR_DRIVER_UNINITIALIZED`: *`AMDTPwrProfileInitialize()`* function was neither called nor successful
- `AMDT_ERROR_TIMER_NOT_SET`: Sampling timer was not set
- `AMDT_ERROR_COUNTERS_NOT_ENABLED`: No counter enabled for collecting profile data
- `AMDT_ERROR_PROFILE_ALREADY_STARTED`: Profile is already started
- `AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED`: Previous session was not closed
- `AMDT_ERROR_BIOS_VERSION_NOT_SUPPORTED`: BIOS needs to be upgraded
- `AMDT_ERROR_FAIL`: An internal error occurred
- `AMDT_ERROR_ACCESSDENIED`: Profiler is busy, currently not accessible

1.2.8 AMDTPwrReadAllEnabledCounters

This API will read all the counters that are enabled. This can return an array of {CounterID, Float-Value}. If there are no new samples, this API will return AMDT_ERROR_PROFILE_DATA_NOT_AVAILABLE and pNumOfSamples will point to value of zero. If there are new samples, this API will return AMDT_STATUS_OK and pNumOfSamples will point to value greater than zero.

AMDTResult **`AMDTPwrReadAllEnabledCounters`**(AMDTUInt32 **pNumOfSamples*, *AMDTPwrSample* ***ppData*)

Parameters

- `pNumOfSamples`: Number of sample based on the AMDTPwrSetSampleValueOption() set
- `ppData`: Processed profile data. No need to allocate or free the memory data is valid till we call this API next time

Returns

- `AMDT_STATUS_OK`: On Success
- `AMDT_ERROR_INVALIDARG`: NULL pointer was passed as pNumSamples of ppData parameters

- `AMDT_ERROR_DRIVER_UNINITIALIZED`: *`AMDTPwrProfileInitialize()`* function was neither called nor successful
- `AMDT_ERROR_PROFILE_NOT_STARTED`: Profile is not started
- `AMDT_ERROR_PROFILE_DATA_NOT_AVAILABLE`: Profile data is not yet available
- `AMDT_ERROR_OUTOFMEMORY`: Memory not available
- `AMDT_ERROR_SMU_ACCESS_FAILED`: One of the configured SMU data accessible
- `AMDT_ERROR_FAIL`: An internal error occurred

1.2.9 AMDTPwrStopProfiling

This APIs will stop the profiling run which was started by *`AMDTPwrStartProfiling()`* function call.

AMDTResult **`AMDTPwrStopProfiling`**`()`

Returns

- `AMDT_STATUS_OK`: On Success
- `AMDT_ERROR_DRIVER_UNINITIALIZED`: *`AMDTPwrProfileInitialize()`* function was neither called nor successful
- `AMDT_ERROR_PROFILE_NOT_STARTED`: Profile is not started
- `AMDT_ERROR_FAIL`: An internal error occurred
- `AMDT_STATUS_OK`: On Success
- `AMDT_ERROR_FAIL`: An internal error occurred
- `AMDT_ERROR_DRIVER_UNINITIALIZED`: *`AMDTPwrProfileInitialize()`* function was neither called nor successful

1.2.10 AMDTPwrProfileClose

This API will close the power profiler and unregister driver and cleanup all memory allocated during *`AMDTPwrProfileInitialize()`*.

AMDTResult **`AMDTPwrProfileClose`**`()`

Returns

- `AMDT_STATUS_OK`: On Success
- `AMDT_ERROR_FAIL`: An internal error occurred
- `AMDT_ERROR_DRIVER_UNINITIALIZED`: *`AMDTPwrProfileInitialize()`* function was neither called nor successful

1.3 Data Types

1.3.1 AMDTPwrProfileMode

enum `AMDTPwrProfileMode`

Following power profile modes are supported.

- `AMDT_PWR_PROFILE_MODE_ONLINE` : Counter values are collected at every specified sampling interval.
- `AMDT_PWR_MODE_INSTANT_COUNTER` : Counter values are collected instantly.

Note: AMDT_PWR_MODE_INSTANT_COUNTER mode is supported only for AMD Family 17h Model 10h based processor family

1.3.2 AMDTCounter

enum `AMDTCounter`

Following power profile counters are supported for *`AMDTPwrGetCounterId()`*.

- `AMD_PWR_SOCKET_POWER` : Socket Power
- `AMD_PWR_SOCKET_STAPM_LIMIT` : Socket Stapm Limit
- `AMD_PWR_SOCKET_PPT_FAST_LIMIT`: Fast PPT Limit
- `AMD_PWR_SOCKET_PPT_SLOW_LIMIT`: Slow PPT Limit

1.3.3 AMDTPwrUnit

enum `AMDTPwrUnit`

Following are the various unit types for the output values for the counter types.

- `AMDT_PWR_UNIT_TYPE_COUNT` : Count index
- `AMDT_PWR_UNIT_TYPE_PERCENT` : Percentage
- `AMDT_PWR_UNIT_TYPE_RATIO` : Ratio
- `AMDT_PWR_UNIT_TYPE_MILLI_SECOND` : Time in milli seconds
- `AMDT_PWR_UNIT_TYPE_JOULE` : Energy consumption
- `AMDT_PWR_UNIT_TYPE_WATT` : Power consumption
- `AMDT_PWR_UNIT_TYPE_VOLT` : Voltage
- `AMDT_PWR_UNIT_TYPE_MILLI_AMPERE` : Current
- `AMDT_PWR_UNIT_TYPE_MEGA_HERTZ` : Frequency
- `AMDT_PWR_UNIT_TYPE_CENTIGRADE` : Temperature

1.3.4 AMDTResult

type `AMDTResult`

```
typedef unsigned int AMDTResult
```

1.3.5 AMDTPwrCounterDesc

type `AMDTPwrCounterDesc`

struct *AMDTPwrCounterDesc* encapsulate details of a supported power counter and its associated device.

Data Members

- `AMDTUInt32 m_counterID` : Counter index
- `AMDTUInt32 m_deviceId` : Device Id
- `AMDTDeviceType m_devType` : Device type- Package/Die/Compute unit/Core/dGPU
- `AMDTUInt32 m_devInstanceId` : Device instance id within the device type
- `char *m_description` : Name of the counter
- `char *m_name` : Description of the counter
- `AMDTPwrCategory m_category` : Power/Frequency/Temperature
- `AMDTPwrAggregation m_aggregation` : Single/Histogram/Cumulative
- `AMDTPwrUnit m_units` : Seconds/MHz/Joules/Watts/Volt/Ampere
- `AMDTUInt32 m_parentCounterId` : If the counter has some child counters

1.3.6 AMDTPwrSample

type `AMDTPwrSample`

struct *AMDTPwrSample* encapsulate output sample with timestamp and the counter values for all the enabled counters.

Data Members

- `AMDTPwrSystemTime m_systemTime` : Start time of Profiling
- `AMDTUInt64 m_elapsedTimeMs` : Elapsed time in milliseconds - relative to the start time of the profile
- `AMDTUInt64 m_recordId` : Record id
- `AMDTUInt32 m_numOfCounter` : Number of counter values available
- `AMDTPwrCounterValue *m_counterValues` : List of counter values

1.3.7 AMDTPwrSystemTime

type `AMDTPwrSystemTime`

struct *AMDTPwrSystemTime* represents the system time in second and milliseconds

Data Members

- `AMDTUInt64 m_second` : Seconds
- `AMDTUInt64 m_microSecond` : Milliseconds

1.3.8 AMDTPwrCounterValue

type `AMDTPwrCounterValue`

struct *AMDTPwrCounterValue* represents a counter id and its value

Data Members

```
AMDTUInt32      m_counterID;      // Counter index
AMDTUInt32      m_valueCnt;       // Number of value for this counter
union
{
    AMDTFloat32 m_data;           // Counter value
    AMDTFloat32 *m_pData;         // Pointer to the multi value array
}
```

1.4 Examples

```
//==============================================================
// (c) 2017 Advanced Micro Devices, Inc.
//
/// \author AMDuProf Developer Tools
/// \brief  Example program using the AMDTPowerProfile APIs.
//
//==============================================================

// - Start the profiling
// - Periodically read the counter values and report till the user has
//   requested to stop

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
#include <string.h>
#ifdef __linux__
    #include <unistd.h>
#endif

#include <AMDTPowerProfileApi.h>

void CollectAllCounters()
{
    AMDTResult hResult = AMDT_STATUS_OK;

    // Initialize online mode
    hResult = AMDTPwrProfileInitialize(AMDT_PWR_MODE_TIMELINE_ONLINE);
    // --- Handle the error

    // Configure the profile run
    //   1. Get the supported counters
    //   2. Enable all the counter
    //   3. Set the timer configuration

    // 1. Get the supported counter details
    AMDTUInt32 nbrCounters = 0;
    AMDTPwrCounterDesc* pCounters = nullptr;

    hResult = AMDTPwrGetSupportedCounters(&nbrCounters, &pCounters);
    assert(AMDT_STATUS_OK == hResult);

    AMDTPwrCounterDesc* pCurrCounter = pCounters;

    for (AMDTUInt32 cnt = 0; cnt < nbrCounters; cnt++, pCurrCounter++)
    {
        if (nullptr != pCurrCounter)
```

```
            {
                // Enable all the counters
                hResult = AMDTPwrEnableCounter(pCurrCounter->m_counterID);
                assert(AMDT_STATUS_OK == hResult);
            }
        }

        // Set the timer configuration
        AMDTUInt32 samplingInterval = 100;      // in milliseconds
        AMDTUInt32 profilingDuration = 10;      // in seconds

        hResult = AMDTPwrSetTimerSamplingPeriod(samplingInterval);
        assert(AMDT_STATUS_OK == hResult);

        // Start the Profile Run
        hResult = AMDTPwrStartProfiling();
        assert(AMDT_STATUS_OK == hResult);

        // Collect and report the counter values periodically
        //   1. Take the snapshot of the counter values
        //   2. Read the counter values
        //   3. Report the counter values

        volatile bool isProfiling = true;
        bool stopProfiling = false;
        AMDTUInt32 nbrSamples = 0;

        while (isProfiling)
        {
            // sleep for refresh duration - at least equivalent to the
            // sampling interval specified
#if defined ( WIN32 )
            // Windows
            Sleep(samplingInterval);
#else
            // Linux
            usleep(samplingInterval * 1000);
#endif

            // read all the counter values
            AMDTPwrSample* pSampleData = nullptr;

            hResult = AMDTPwrReadAllEnabledCounters(&nbrSamples, &pSampleData);

            if (AMDT_STATUS_OK != hResult)
            {
                continue;
            }

            if (nullptr != pSampleData)
            {
                // iterate over all the samples and report the sampled counter values
                for (AMDTUInt32 idx = 0; idx < nbrSamples; idx++)
                {
                    // Iterate over the sampled counter values and print
                    for (unsigned int i = 0; i < pSampleData[idx].m_numOfCounter; i++)
                    {
                        if (nullptr != pSampleData[idx].m_counterValues)
                        {
                            AMDUInt32 id = 0;
                            id = pSampleData[idx].m_counterValues->m_counterID;
                            // Get the counter descriptor to print the counter
                            // name
                            AMDTPwrCounterDesc counterDesc;
                            AMDTPwrGetCounterDesc(id, &counterDesc);

                            fprintf(stdout, "%s : %f ",
                                                                    counterDesc.m_name,
                                    pSampleData[idx].m_counterValues->m_data);

                            pSampleData[idx].m_counterValues++;
                        }
                    } // iterate over the sampled counters

                    fprintf(stdout, "\n");
                } // iterate over all the samples collected

                // check if we exceeded the profile duration
                if ((profilingDuration > 0)
                    && (pSampleData->m_elapsedTimeMs >= (profilingDuration * 1000)))
                {
                    stopProfiling = true;
                }

                if (stopProfiling)
                {
                    // stop the profiling
                    hResult = AMDTPwrStopProfiling();
                    assert(AMDT_STATUS_OK == hResult);
                    isProfiling = false;
                }
            }
        }

    // Close the profiler
    hResult = AMDTPwrProfileClose();
    assert(AMDT_STATUS_OK == hResult);
}

int main()
{
    AMDTResult hResult = AMDT_STATUS_OK;
    CollectAllCounters();
    return hResult;
}
```

1.5 How to use API Library

Example code must use AMDTPowerProfileAPI.dll to compile and run the example program. We must need to make sure drivers are up and running.

To build and execute a example application, following steps should be performed on Linux machine.

1. **Example CollectAllCounters is located at `<AMDuProf-install-dir>/Examples/CollectAllCounters`**

   ```
   $ cd <AMDuProf-install-dir>/Examples/CollectAllCounters
   ```

2. **Set LD_LIBRARY_PATH**

   ```
   $ export LD_LIBRARY_PATH=<AMDuProf-install-dir>/bin
   ```

3. **Compile application code**

   ```
   $ g++ -O -std=c++11 CollectAllCounters.cpp
   ```

```
-I<AMDuProf-install-dir>/include
-l AMDPowerProfileAPI -L<AMDuProf-install-dir>/bin
-Wl,-rpath <AMDuProf-install-dir>/bin -o CollectAllCounters
```

4. **Execute**

```
$ ./CollectAllCounters
```