Intel Performance Counter Monitor V2.6 Reference Manual

Generated by Doxygen 1.8.8

Mon Aug 25 2014 19:11:15 Generated by tangloner

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

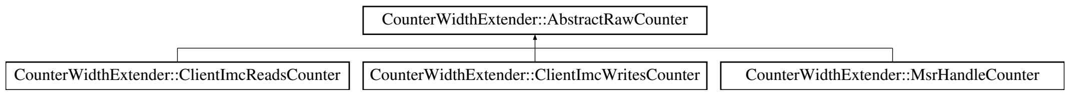
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 CounterWidthExtender::AbstractRawCounter Struct Reference

Inheritance diagram for CounterWidthExtender::AbstractRawCounter:


CounterWidthExtender::AbstractRawCounter
CounterWidthExtender::ClientImcReadsCounter
CounterWidthExtender::ClientImcWritesCounter
CounterWidthExtender::MsrHandleCounter

Public Member Functions

- virtual uint64 **operator()** ()=0

The documentation for this struct was generated from the following file:

- width_extender.h

4.2 AsynchronCounterState Class Reference

Public Member Functions

- uint32 **getNumCores** ()
- uint32 **getNumSockets** ()
- uint32 **getQPILinksPerSocket** ()
- uint32 **getSocketId** (uint32 c)
- template<typename T , T func>
  T get (uint32 core)
- template<typename T , T func>
  T get (int param, uint32 core)
- template<typename T , T func>
  T getSocket (uint32 socket)
- template<typename T , T func>
  T getSocket (int param, uint32 socket)
- template<typename T , T func>
  T getSocket (uint32 socket, uint32 param)
- template<typename T , T func>
  T getSystem ()
- template<typename T , T func>
  T getSystem (int param)

Friends

- void ∗ **UpdateCounters** (void ∗)

The documentation for this class was generated from the following file:

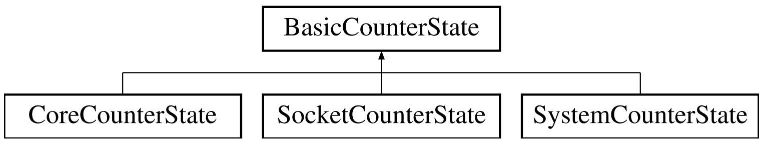
- **cpuasynchcounter.h**

4.3 BasicCounterState Class Reference

Basic core counter state.

`#include <cpucounters.h>`

Inheritance diagram for BasicCounterState:



Public Member Functions

- **BasicCounterState** & **operator+=** (const **BasicCounterState** &o)
- int32 **getThermalHeadroom** () const

  Returns current thermal headroom below TjMax.

Protected Member Functions

- void **readAndAggregate** (**MsrHandle** ∗)

Protected Attributes

- uint64 **InstRetiredAny**
- uint64 **CpuClkUnhaltedThread**
- uint64 **CpuClkUnhaltedRef**
- union {
  uint64 **L3Miss**
  uint64 **Event0**
  uint64 **ArchLLCMiss**
  };
- union {
  uint64 **L3UnsharedHit**
  uint64 **Event1**
  uint64 **ArchLLCRef**
  };
- union {
  uint64 **L2HitM**
  uint64 **Event2**
  };

- union {
  uint64 **L2Hit**
  uint64 **Event3**
  };
- uint64 **InvariantTSC**
- uint64 **CStateResidency** [PCM::MAX_C_STATE+1]
- int32 **ThermalHeadroom**

Friends

- class **PCM**
- template<class CounterStateType >
  double **getExecUsage** (const CounterStateType &before, const CounterStateType &after)

  Computes average number of retired instructions per time intervall.
- template<class CounterStateType >
  double **getIPC** (const CounterStateType &before, const CounterStateType &after)

  Computes average number of retired instructions per core cycle (IPC)
- template<class CounterStateType >
  double **getAverageFrequency** (const CounterStateType &before, const CounterStateType &after)

  Computes average core frequency also taking Intel Turbo Boost technology into account.
- template<class CounterStateType >
  double **getActiveAverageFrequency** (const CounterStateType &before, const CounterStateType &after)

  Computes average core frequency when not in powersaving C0-state (also taking Intel Turbo Boost technology into account)
- template<class CounterStateType >
  double **getCyclesLostDueL3CacheMisses** (const CounterStateType &before, const CounterStateType &after)

  Estimates how many core cycles were potentially lost due to L3 cache misses.
- template<class CounterStateType >
  double **getCyclesLostDueL2CacheMisses** (const CounterStateType &before, const CounterStateType &after)

  Estimates how many core cycles were potentially lost due to missing L2 cache but still hitting L3 cache.
- template<class CounterStateType >
  double **getRelativeFrequency** (const CounterStateType &before, const CounterStateType &after)

  Computes average core frequency also taking Intel Turbo Boost technology into account.
- template<class CounterStateType >
  double **getActiveRelativeFrequency** (const CounterStateType &before, const CounterStateType &after)

  Computes average core frequency when not in powersaving C0-state (also taking Intel Turbo Boost technology into account)
- template<class CounterStateType >
  double **getL2CacheHitRatio** (const CounterStateType &before, const CounterStateType &after)

  Computes L2 cache hit ratio.
- template<class CounterStateType >
  double **getL3CacheHitRatio** (const CounterStateType &before, const CounterStateType &after)

  Computes L3 cache hit ratio.
- template<class CounterStateType >
  uint64 **getL3CacheMisses** (const CounterStateType &before, const CounterStateType &after)

  Computes number of L3 cache misses.
- template<class CounterStateType >
  uint64 **getL2CacheMisses** (const CounterStateType &before, const CounterStateType &after)

  Computes number of L2 cache misses.
- template<class CounterStateType >
  uint64 **getL2CacheHits** (const CounterStateType &before, const CounterStateType &after)

Computes number of L2 cache hits.

- template<class CounterStateType >
  uint64 **getCycles** (const CounterStateType &before, const CounterStateType &after)

  Computes the number core clock cycles when signal on a specific core is running (not halted)
- template<class CounterStateType >
  uint64 **getInstructionsRetired** (const CounterStateType &before, const CounterStateType &after)

  Computes the number of retired instructions.
- template<class CounterStateType >
  uint64 **getCycles** (const CounterStateType &now)

  Computes the number executed core clock cycles.
- template<class CounterStateType >
  uint64 **getInstructionsRetired** (const CounterStateType &now)

  Computes the number of retired instructions.
- template<class CounterStateType >
  uint64 **getL3CacheHitsNoSnoop** (const CounterStateType &before, const CounterStateType &after)

  Computes number of L3 cache hits where no snooping in sibling L2 caches had to be done.
- template<class CounterStateType >
  uint64 **getL3CacheHitsSnoop** (const CounterStateType &before, const CounterStateType &after)

  Computes number of L3 cache hits where snooping in sibling L2 caches had to be done.
- template<class CounterStateType >
  uint64 **getL3CacheHits** (const CounterStateType &before, const CounterStateType &after)

  Computes total number of L3 cache hits.
- template<class CounterStateType >
  uint64 **getNumberOfCustomEvents** (int32 eventCounterNr, const CounterStateType &before, const CounterStateType &after)

  Returns the number of occured custom core events.
- template<class CounterStateType >
  uint64 **getInvariantTSC** (const CounterStateType &before, const CounterStateType &after)

  Computes number of invariant time stamp counter ticks.
- template<class CounterStateType >
  uint64 **getRefCycles** (const CounterStateType &before, const CounterStateType &after)

  Computes the number of reference clock cycles while clock signal on the core is running.
- template<class CounterStateType >
  double **getCoreCStateResidency** (int state, const CounterStateType &before, const CounterStateType &after)

  Computes residency in the core C-state.

4.3.1 Detailed Description

Basic core counter state.

Intended only for derivation, but not for the direct use

4.3.2 Friends And Related Function Documentation

4.3.2.1 template<class CounterStateType > double getActiveAverageFrequency (const CounterStateType & *before*, const CounterStateType & *after*) [friend]

Computes average core frequency when not in powersaving C0-state (also taking Intel Turbo Boost technology into account)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

frequency in Hz

4.3.2.2 template<class CounterStateType > double getActiveRelativeFrequency (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes average core frequency when not in powersaving C0-state (also taking Intel Turbo Boost technology into account)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Fraction of nominal frequency (if >1.0 then Turbo was working during the measurement)

4.3.2.3 template<class CounterStateType > double getAverageFrequency (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes average core frequency also taking Intel Turbo Boost technology into account.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

frequency in Hz

4.3.2.4 template<class CounterStateType > double getCoreCStateResidency (int *state,* const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes residency in the core C-state.

Parameters

state	C-state
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

residence ratio (0..1): 0 - 0%, 1.0 - 100%

4.3.2.5 template<class CounterStateType > uint64 getCycles (const CounterStateType & *before*, const CounterStateType & *after*) [friend]

Computes the number core clock cycles when signal on a specific core is running (not halted)

Returns number of used cycles (halted cyles are not counted). The counter does not advance in the following conditions:

- an ACPI C-state is other than C0 for normal operation
- HLT
- STPCLK+ pin is asserted
- being throttled by TM1
- during the frequency switching phase of a performance state transition

The performance counter for this event counts across performance state transitions using different core clock frequencies

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

number core clock cycles

4.3.2.6 template<class CounterStateType > uint64 getCycles (const CounterStateType & *now*) [friend]

Computes the number executed core clock cycles.

Returns number of used cycles (halted cyles are not counted).

Parameters

now	Current CPU counter state

Returns

number core clock cycles

4.3.2.7 template<class CounterStateType > double getCyclesLostDueL2CacheMisses (const CounterStateType & *before*, const CounterStateType & *after*) [friend]

Estimates how many core cycles were potentially lost due to missing L2 cache but still hitting L3 cache.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)
Currently not supported on Intel(R) Atom(tm) processor

Returns

ratio that is usually beetween 0 and 1 ; in some cases could be >1.0 due to a lower access latency estimation

4.3.2.8 template<class CounterStateType > double getCyclesLostDueL3CacheMisses (const CounterStateType & *before*, const CounterStateType & *after*) `[friend]`

Estimates how many core cycles were potentially lost due to L3 cache misses.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

ratio that is usually beetween 0 and 1 ; in some cases could be >1.0 due to a lower memory latency estimation

4.3.2.9 template<class CounterStateType > double getExecUsage (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Computes average number of retired instructions per time intervall.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

usage

4.3.2.10 template<class CounterStateType > uint64 getInstructionsRetired (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Computes the number of retired instructions.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

number of retired instructions

4.3.2.11 template<class CounterStateType > uint64 getInstructionsRetired (const CounterStateType & *now*) [friend]

Computes the number of retired instructions.

Parameters

now	Current CPU counter state

Returns

number of retired instructions

4.3.2.12 template<class CounterStateType > uint64 getInvariantTSC (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Computes number of invariant time stamp counter ticks.

This counter counts irrespectively of C-, P- or T-states

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

number of time stamp counter ticks

4.3.2.13 template<class CounterStateType > double getIPC (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes average number of retired instructions per core cycle (IPC)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

IPC

4.3.2.14 template<class CounterStateType > double getL2CacheHitRatio (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes L2 cache hit ratio.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

value between 0 and 1

4.3.2.15 template<class CounterStateType > uint64 getL2CacheHits (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes number of L2 cache hits.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of hits

4.3.2.16 template$<$class CounterStateType $>$ uint64 getL2CacheMisses (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes number of L2 cache misses.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of misses

4.3.2.17 template<class CounterStateType > double getL3CacheHitRatio (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes L3 cache hit ratio.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

value between 0 and 1

4.3.2.18 template<class CounterStateType > uint64 getL3CacheHits (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes total number of L3 cache hits.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of hits

4.3.2.19 template<class CounterStateType > uint64 getL3CacheHitsNoSnoop (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes number of L3 cache hits where no snooping in sibling L2 caches had to be done.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of hits

4.3.2.20 template< class CounterStateType > uint64 getL3CacheHitsSnoop (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes number of L3 cache hits where snooping in sibling L2 caches had to be done.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of hits

4.3.2.21 template< class CounterStateType > uint64 getL3CacheMisses (const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Computes number of L3 cache misses.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of misses

4.3.2.22 template< class CounterStateType > uint64 getNumberOfCustomEvents (int32 *eventCounterNr,* const CounterStateType & *before,* const CounterStateType & *after*) `[friend]`

Returns the number of occured custom core events.

Read number of events programmed with the `CUSTOM_CORE_EVENTS`

Parameters

eventCounterNr	Event/counter number (value from 0 to 3)
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Number of bytes

4.3.2.23 template< class CounterStateType > uint64 getRefCycles (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Computes the number of reference clock cycles while clock signal on the core is running.

The reference clock operates at a fixed frequency, irrespective of core frequency changes due to performance state transitions. See Intel(r) Software Developer's Manual for more details

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

number core clock cycles

4.3.2.24 template< class CounterStateType > double getRelativeFrequency (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Computes average core frequency also taking Intel Turbo Boost technology into account.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Fraction of nominal frequency

The documentation for this class was generated from the following files:

- **cpucounters.h**
- cpucounters.cpp

4.4 BecktonUncorePMUCNTCTLRegister Struct Reference

Public Attributes

- union {
  struct {
    uint64 **en**: 1
    uint64 **pmi_en**: 1
    uint64 **count_mode**: 2
    uint64 **storage_mode**: 2

uint64 **wrap_mode**: 1
uint64 **flag_mode**: 1
uint64 **rsv1**: 1
uint64 **inc_sel**: 5
uint64 **rsv2**: 5
uint64 **set_flag_sel**: 3
} **fields**
uint64 **value**
};

The documentation for this struct was generated from the following file:

- **types.h**

4.5 BecktonUncorePMUZDPCTLFVCRegister Struct Reference

Public Attributes

- union {
  struct {
  uint64 **fvid**: 5
  uint64 **bcmd**: 3
  uint64 **resp**: 3
  uint64 **evnt0**: 3
  uint64 **evnt1**: 3
  uint64 **evnt2**: 3
  uint64 **evnt3**: 3
  uint64 **pbox_init_err**: 1
  } **fields**
  struct {
  uint64 **fvid**: 6
  uint64 **bcmd**: 3
  uint64 **resp**: 3
  uint64 **evnt0**: 3
  uint64 **evnt1**: 3
  uint64 **evnt2**: 3
  uint64 **evnt3**: 3
  uint64 **pbox_init_err**: 1
  } **fields_wsm**
  uint64 **value**
  };

The documentation for this struct was generated from the following file:

- **types.h**

4.6 ClientBW Class Reference

Public Member Functions

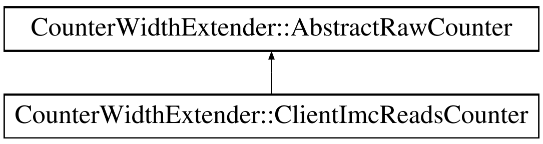
- uint64 **getImcReads** ()
- uint64 **getImcWrites** ()

The documentation for this class was generated from the following file:

- **client_bw.h**

4.7 CounterWidthExtender::ClientImcReadsCounter Struct Reference

Inheritance diagram for CounterWidthExtender::ClientImcReadsCounter:


CounterWidthExtender::AbstractRawCounter
CounterWidthExtender::ClientImcReadsCounter

Public Member Functions

- **ClientImcReadsCounter** (**ClientBW** ∗clientBW_)
- uint64 **operator()** ()

Public Attributes

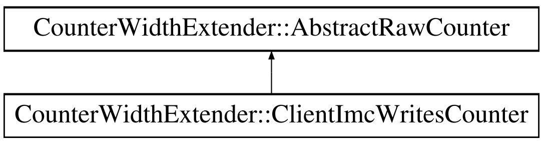
- **ClientBW** ∗ **clientBW**

The documentation for this struct was generated from the following file:

- width_extender.h

4.8 CounterWidthExtender::ClientImcWritesCounter Struct Reference

Inheritance diagram for CounterWidthExtender::ClientImcWritesCounter:


CounterWidthExtender::AbstractRawCounter
CounterWidthExtender::ClientImcWritesCounter

Public Member Functions

- **ClientImcWritesCounter** (**ClientBW** ∗clientBW_)
- uint64 **operator()** ()

Public Attributes

- **ClientBW** ∗ **clientBW**

The documentation for this struct was generated from the following file:

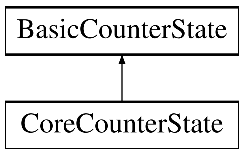
- width_extender.h

4.9 CoreCounterState Class Reference

(Logical) core-wide counter state

```
#include <cpucounters.h>
```

Inheritance diagram for CoreCounterState:



Friends

- class **PCM**

Additional Inherited Members

4.9.1 Detailed Description

(Logical) core-wide counter state

The documentation for this class was generated from the following file:

- **cpucounters.h**

4.10 CounterWidthExtender Class Reference

Classes

- struct **AbstractRawCounter**
- struct **ClientImcReadsCounter**
- struct **ClientImcWritesCounter**
- struct **MsrHandleCounter**

Public Member Functions

- **CounterWidthExtender** (**AbstractRawCounter** ∗raw_counter_)
- uint64 **read** ()

The documentation for this class was generated from the following file:

- width_extender.h

4.11 PCM::CustomCoreEventDescription Struct Reference

Custom Core event description.

```
#include <cpucounters.h>
```

Public Attributes

- int32 **event_number**
- int32 **umask_value**

4.11.1 Detailed Description

Custom Core event description.

See "Intel 64 and IA-32 Architectures Software Developers Manual Volume 3B: System Programming Guide, Part 2" for the concrete values of the data structure fields, e.g. Appendix A.2 "Performance Monitoring Events for Intel(r) Core(tm) Processor Family and Xeon Processor Family"

The documentation for this struct was generated from the following file:

- **cpucounters.h**

4.12 EventSelectRegister Struct Reference

Public Attributes

- union {
    struct {
        uint64 **event_select**: 8
        uint64 **umask**: 8
        uint64 **usr**: 1
        uint64 **os**: 1
        uint64 **edge**: 1
        uint64 **pin_control**: 1
        uint64 **apic_int**: 1
        uint64 **any_thread**: 1
        uint64 **enable**: 1
        uint64 **invert**: 1
        uint64 **cmask**: 8
        uint64 **in_tx**: 1
        uint64 **in_txcp**: 1
        uint64 **reservedX**: 30
    } **fields**
    uint64 **value**
  };

The documentation for this struct was generated from the following file:

- **types.h**

4.13 PCM::ExtendedCustomCoreEventDescription Struct Reference

Extended custom core event description.

```
#include <cpucounters.h>
```

Public Attributes

- **FixedEventControlRegister** $*$ **fixedCfg**

- uint32 **nGPCounters**
- **EventSelectRegister** ∗ **gpCounterCfg**
- uint64 **OffcoreResponseMsrValue** [2]

4.13.1 Detailed Description

Extended custom core event description.

In contrast to **CustomCoreEventDescription** (p. 22) supports configuration of all fields.

See "Intel 64 and IA-32 Architectures Software Developers Manual Volume 3B: System Programming Guide, Part 2" for the concrete values of the data structure fields, e.g. Appendix A.2 "Performance Monitoring Events for Intel(r) Core(tm) Processor Family and Xeon Processor Family"

The documentation for this struct was generated from the following file:

- **cpucounters.h**

4.14 FixedEventControlRegister Struct Reference

Public Attributes

```
• union {
    struct {
      uint64 os0: 1
      uint64 usr0: 1
      uint64 any_thread0: 1
      uint64 enable_pmi0: 1
      uint64 os1: 1
      uint64 usr1: 1
      uint64 any_thread1: 1
      uint64 enable_pmi1: 1
      uint64 os2: 1
      uint64 usr2: 1
      uint64 any_thread2: 1
      uint64 enable_pmi2: 1
      uint64 reserved1: 52
    } fields
    uint64 value
  };
```

The documentation for this struct was generated from the following file:

- **types.h**

4.15 MCFGHeader Struct Reference

Public Member Functions

- unsigned **nrecords** () const
- void **print** ()

Public Attributes

- char **signature** [4]
- unsigned **length**
- unsigned char **revision**
- unsigned char **checksum**
- char **OEMID** [6]
- char **OEMTableID** [8]
- unsigned **OEMRevision**
- unsigned **creatorID**
- unsigned **creatorRevision**
- char **reserved** [8]

The documentation for this struct was generated from the following file:

- **types.h**

4.16 MCFGRecord Struct Reference

Public Member Functions

- void **print** ()

Public Attributes

- unsigned long long **baseAddress**
- unsigned short **PCISegmentGroupNumber**
- unsigned char **startBusNumber**
- unsigned char **endBusNumber**
- char **reserved** [4]

The documentation for this struct was generated from the following file:

- **types.h**

4.17 MsrHandle Class Reference

Public Member Functions

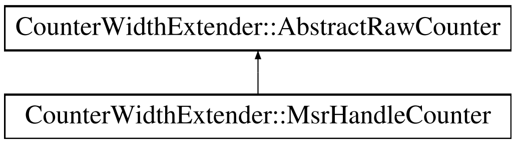
- **MsrHandle** (uint32 cpu)
- int32 **read** (uint64 msr_number, uint64 ∗value)
- int32 **write** (uint64 msr_number, uint64 value)
- uint32 **getCoreId** ()

The documentation for this class was generated from the following files:

- **msr.h**
- msr.cpp

4.18 CounterWidthExtender::MsrHandleCounter Struct Reference

Inheritance diagram for CounterWidthExtender::MsrHandleCounter:


CounterWidthExtender::AbstractRawCounter
CounterWidthExtender::MsrHandleCounter

Public Member Functions

- **MsrHandleCounter** (**MsrHandle** ∗msr_, uint64 msr_addr_)
- uint64 **operator()** ()

Public Attributes

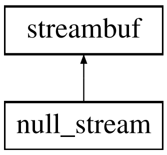
- **MsrHandle** ∗ **msr**
- uint64 **msr_addr**

The documentation for this struct was generated from the following file:

- width_extender.h

4.19 null_stream Struct Reference

Inheritance diagram for null_stream:



Public Member Functions

- void **overflow** (char)

The documentation for this struct was generated from the following file:

- utils.h

4.20 PCIeCounterState Class Reference

Friends

- class **PCM**
- uint64 **getNumberOfEvents** (**PCIeCounterState** before, **PCIeCounterState** after)

    Returns the raw count of PCIe events.

4.20.1 Friends And Related Function Documentation

4.20.1.1 uint64 getNumberOfEvents (**PCIeCounterState** *before*, **PCIeCounterState** *after*) `[friend]`

Returns the raw count of PCIe events.

Parameters

before	PCIe counter state before the experiment
after	PCIe counter state after the experiment

The documentation for this class was generated from the following file:

- **cpucounters.h**

4.21 PCIeEvents_t Struct Reference

Public Attributes

- uint64 **PCIePRd**
- uint64 **PCIeRdCur**
- uint64 **PCIeNSRd**
- uint64 **PCIeWiLF**
- uint64 **PCIeItoM**
- uint64 **PCIeNSWr**
- uint64 **PCIeNSWrF**

The documentation for this struct was generated from the following file:

- **pcm-pcie.cpp**

4.22 PciHandle Class Reference

Public Member Functions

- **PciHandle** (uint32 groupnr_, uint32 bus_, uint32 device_, uint32 function_)
- int32 **read32** (uint64 offset, uint32 ∗value)
- int32 **write32** (uint64 offset, uint32 value)
- int32 **read64** (uint64 offset, uint64 ∗value)
- int32 **write64** (uint64 offset, uint64 value)

Static Public Member Functions

- static bool **exists** (uint32 bus_, uint32 device_, uint32 function_)

The documentation for this class was generated from the following files:

- **pci.h**
- pci.cpp

4.23 PciHandleM Class Reference

Public Member Functions

- **PciHandleM** (uint32 bus_, uint32 device_, uint32 function_)
- int32 **read32** (uint64 offset, uint32 ∗value)
- int32 **write32** (uint64 offset, uint32 value)
- int32 **read64** (uint64 offset, uint64 ∗value)
- int32 **write64** (uint64 offset, uint64 value)

Static Public Member Functions

- static bool **exists** (uint32 bus_, uint32 device_, uint32 function_)

The documentation for this class was generated from the following file:

- **pci.h**

4.24 PciHandleMM Class Reference

Public Member Functions

- **PciHandleMM** (uint32 groupnr_, uint32 bus_, uint32 device_, uint32 function_)
- int32 **read32** (uint64 offset, uint32 ∗value)
- int32 **write32** (uint64 offset, uint32 value)
- int32 **read64** (uint64 offset, uint64 ∗value)
- int32 **write64** (uint64 offset, uint64 value)

Static Public Member Functions

- static bool **exists** (uint32 bus_, uint32 device_, uint32 function_)

The documentation for this class was generated from the following files:

- **pci.h**
- pci.cpp

4.25 PCM Class Reference

CPU Performance Monitor.

```
#include <cpucounters.h>
```

Classes

- struct **CustomCoreEventDescription**

  Custom Core event description.
- struct **ExtendedCustomCoreEventDescription**

  Extended custom core event description.

Public Types

- enum { **MAX_C_STATE** = 10 }
- enum **ProgramMode** { **DEFAULT_EVENTS** = 0, **CUSTOM_CORE_EVENTS** = 1, **EXT_CUSTOM_CORE**↩ **_EVENTS** = 2, **INVALID_MODE** }

  *Mode of programming (parameter in the **program()** (p. 38) method)*
- enum **ErrorCode** { **Success** = 0, **MSRAccessDenied** = 1, **PMUBusy** = 2, **UnknownError** }

  Return codes (e.g. for program(..) method)
- enum **SupportedCPUModels** {
  NEHALEM_EP = 26, **NEHALEM** = 30, **ATOM** = 28, **ATOM_2** = 53,
  ATOM_CENTERTON = 54, **ATOM_BAYTRAIL** = 55, **ATOM_AVOTON** = 77, **CLARKDALE** = 37,
  WESTMERE_EP = 44, **NEHALEM_EX** = 46, **WESTMERE_EX** = 47, **SANDY_BRIDGE** = 42,
  JAKETOWN = 45, **IVY_BRIDGE** = 58, **HASWELL** = 60, **HASWELL_ULT** = 69,
  HASWELL_2 = 70, **IVYTOWN** = 62, **END_OF_MODEL_LIST** }

  Identifiers of supported CPU models.
- enum **PCIeEventCode** {
  PCIePRd = 0x195, **PCIeRdCur** = 0x19E, **PCIeNSRd** = 0x1E4, **PCIeWiLF** = 0x194,
  PCIeItoM = 0x19C, **PCIeNSWr** = 0x1E5, **PCIeNSWrF** = 0x1E6 }

Public Member Functions

- bool **isCoreCStateResidencySupported** (int state)

  Returns true if the specified core C-state residency metric is supported.
- bool **isPackageCStateResidencySupported** (int state)

  Returns true if the specified package C-state residency metric is supported.
- bool **good** ()

  *Checks the status of **PCM** (p. 28) object.*
- const std::string & **getErrorMessage** () const

  Returns the error message.
- **ErrorCode program** (**ProgramMode** mode_=**DEFAULT_EVENTS**, void ∗parameter_=NULL)

  Programs performance counters.
- **ErrorCode programServerUncorePowerMetrics** (int mc_profile, int pcu_profile, int ∗freq_bands=NULL)

  Programs uncore power/energy counters on microarchitectures codename SandyBridge-EP and IvyTown.
- void **freezeServerUncoreCounters** ()

  Freezes uncore event counting (works only on microarchitecture codename SandyBridge-EP and IvyTown)
- void **unfreezeServerUncoreCounters** ()

  Unfreezes uncore event counting (works only on microarchitecture codename SandyBridge-EP and IvyTown)
- **ServerUncorePowerState getServerUncorePowerState** (uint32 socket)

  Reads the power/energy counter state of a socket (works only on microarchitecture codename SandyBridge-EP)
- void **cleanup** ()

  Cleanups resources and stops performance counting.
- void **resetPMU** ()

  Forces PMU reset.
- void **getAllCounterStates** (**SystemCounterState** &systemState, std::vector< **SocketCounterState** > &socketStates, std::vector< **CoreCounterState** > &coreStates)

  Reads all counter states (including system, sockets and cores)
- **SystemCounterState getSystemCounterState** ()

  Reads the counter state of the system.
- **SocketCounterState getSocketCounterState** (uint32 socket)

  Reads the counter state of a socket.
- **CoreCounterState getCoreCounterState** (uint32 core)

  Reads the counter state of a (logical) core.

- uint32 **getNumCores** ()

  Reads number of logical cores in the system.
- uint32 **getNumSockets** ()

  Reads number of sockets (CPUs) in the system.
- uint32 **getThreadsPerCore** ()

  Reads how many hardware threads has a physical core "Hardware thread" is a logical core in a different terminology. If Intel(r) Hyperthreading(tm) is enabled then this function returns 2.
- bool **getSMT** ()

  Checks if SMT (HyperThreading) is enabled.
- uint64 **getNominalFrequency** ()

  Reads the nominal core frequency.
- uint32 **getCPUModel** ()

  Reads CPU model id.
- uint32 **getOriginalCPUModel** ()

  Reads original CPU model id.
- uint32 **getSocketId** (uint32 core_id)

  Determines socket of given core.
- uint64 **getQPILinksPerSocket** () const

  Returns the number of Intel(r) Quick Path Interconnect(tm) links per socket.
- uint32 **getMCPerSocket** () const

  Returns the number of detected integrated memory controllers per socket.
- uint32 **getMCChannelsPerSocket** () const

  Returns the total number of detected memory channels on all integrated memory controllers per socket.
- uint32 **getMaxIPC** () const

  Returns the max number of instructions per cycle.
- uint64 **getTickCount** (uint64 multiplier=1000, uint32 core=0)

  Return TSC timer value in time units.
- uint64 **getTickCountRDTSCP** (uint64 multiplier=1000)

  Return TSC timer value in time units using rdtscp instruction from current core.
- uint64 **getQPILinkSpeed** () const

  Return QPI Link Speed in GBytes/second.
- double **getJoulesPerEnergyUnit** () const

  Returns how many joules are in an internal processor energy unit.
- int32 **getPackageThermalSpecPower** () const

  Returns thermal specification power of the package domain in Watt.
- int32 **getPackageMinimumPower** () const

  Returns minimum power derived from electrical spec of the package domain in Watt.
- int32 **getPackageMaximumPower** () const

  Returns maximum power derived from electrical spec of the package domain in Watt.
- void **disableJKTWorkaround** ()
- void **programPCIeCounters** (const PCIeEventCode event_)

  Program uncore PCIe monitoring event(s)
- **PCIeCounterState getPCIeCounterState** (const uint32 socket_)

  Get the state of PCIe counter(s)
- uint64 **extractCoreGenCounterValue** (uint64 val)
- uint64 **extractCoreFixedCounterValue** (uint64 val)
- uint64 **extractUncoreGenCounterValue** (uint64 val)
- uint64 **extractUncoreFixedCounterValue** (uint64 val)
- const char ∗ **getUArchCodename** ()

  Get a string describing the codename of the processor microarchitecture.
- bool **packageEnergyMetricsAvailable** () const

- bool **dramEnergyMetricsAvailable** () const
- bool **packageThermalMetricsAvailable** () const
- bool **outgoingQPITrafficMetricsAvailable** () const
- bool **qpiUtilizationMetricsAvailable** () const
- bool **memoryTrafficMetricsAvailable** () const
- bool **hasBecktonUncore** () const
- bool **hasPCICFGUncore** () const

Static Public Member Functions

- static **PCM** $*$ **getInstance** ()

  *Returns **PCM** (p. 28) object.*
- static bool **initWinRing0Lib** ()

  Loads and initializes Winring0 third party library for access to processor model specific and PCI configuration registers.
- static std::string **getCPUBrandString** ()

  Get Brand string of processor.

Friends

- class **BasicCounterState**
- class **UncoreCounterState**

4.25.1 Detailed Description

CPU Performance Monitor.

This singleton object needs to be instantiated for each process before accessing counting and measuring routines

4.25.2 Member Enumeration Documentation

4.25.2.1 enum PCM::ProgramMode

Mode of programming (parameter in the **program()** (p. 38) method)

Enumerator

DEFAULT_EVENTS Default choice of events, the additional parameter is not needed and ignored

CUSTOM_CORE_EVENTS Custom set of core events specified in the parameter to the program method. The parameter must be a pointer to array of four **`CustomCoreEventDescription`** (p. 22) values

EXT_CUSTOM_CORE_EVENTS Custom set of core events specified in the parameter to the program method. The parameter must be a pointer to a **`ExtendedCustomCoreEventDescription`** (p. 23) data structure

INVALID_MODE Non-programmed mode

4.25.3 Member Function Documentation

4.25.3.1 void PCM::cleanup ( )

Cleanups resources and stops performance counting.

One needs to call this method when your program finishes or/and you are not going to use the performance counting routines anymore.

4.25.3.2 void PCM::getAllCounterStates (**SystemCounterState** & *systemState,* std::vector< **SocketCounterState** > & *socketStates,* std::vector< **CoreCounterState** > & *coreStates*)

Reads all counter states (including system, sockets and cores)

Parameters

systemState	system counter state (return parameter)
socketStates	socket counter states (return parameter)
coreStates	core counter states (return parameter)

4.25.3.3 CoreCounterState PCM::getCoreCounterState (uint32 *core*)

Reads the counter state of a (logical) core.

Be aware that during the measurement other threads may be scheduled on the same core by the operating system (this is called context-switching). The performance events caused by these threads will be counted as well.

```
\param core core id
\return State of counters in the core
```

Referenced by getCoreCounterState(), and getTickCount().

4.25.3.4 uint32 PCM::getCPUModel ( ) [inline]

Reads CPU model id.

Returns

CPU model ID

Referenced by getCyclesLostDueL2CacheMisses(), getCyclesLostDueL3CacheMisses(), getL2CacheHitRatio(), getL2CacheHits(), getL2CacheMisses(), getL3CacheHitRatio(), getL3CacheHits(), getL3CacheHitsNoSnoop(), getL3CacheHitsSnoop(), getL3CacheMisses(), and ServerPCICFGUncore::ServerPCICFGUncore().

4.25.3.5 const std::string& PCM::getErrorMessage ( ) const [inline]

Returns the error message.

Call this when **good()** (p. 38) returns false, otherwise return an empty string

4.25.3.6 PCM ∗ PCM::getInstance ( ) [static]

Returns **PCM** (p. 28) object.

Returns **PCM** (p. 28) object. If the **PCM** (p. 28) has not been created before than an instance is created. **PCM** (p. 28) is a singleton.

Returns

Pointer to **PCM** (p. 28) object

Referenced by ServerPCICFGUncore::computeQPISpeed(), getActiveAverageFrequency(), getAllIncomingQP↩ ILinkBytes(), getAllOutgoingQPILinkBytes(), getAverageFrequency(), getConsumedJoules(), getCoreCounter↩ State(), getCoreCStateResidency(), getCoreIPC(), getCyclesLostDueL2CacheMisses(), getCyclesLostDueL3↩ CacheMisses(), getDRAMConsumedJoules(), getIncomingQPILinkUtilization(), getL2CacheHitRatio(), getL2↩ CacheHits(), getL2CacheMisses(), getL3CacheHitRatio(), getL3CacheHits(), getL3CacheHitsNoSnoop(), get↩ L3CacheHitsSnoop(), getL3CacheMisses(), getOutgoingQPILinkBytes(), getOutgoingQPILinkUtilization(), get↩ SocketCounterState(), getSocketIncomingQPILinkBytes(), getSystemCounterState(), and getTotalExecUsage().

4.25.3.7 uint32 PCM::getMaxIPC ( ) const [inline]

Returns the max number of instructions per cycle.

Returns

max number of instructions per cycle

4.25.3.8 uint64 PCM::getNominalFrequency ( )

Reads the nominal core frequency.

Returns

Nominal frequency in Hz

Referenced by getActiveAverageFrequency(), getAverageFrequency(), getIncomingQPILinkUtilization(), get↩ OutgoingQPILinkBytes(), getOutgoingQPILinkUtilization(), getTickCount(), and getTickCountRDTSCP().

4.25.3.9 uint32 PCM::getNumCores ( )

Reads number of logical cores in the system.

Returns

Number of logical cores in the system

Referenced by getIncomingQPILinkUtilization(), getOutgoingQPILinkBytes(), and getOutgoingQPILinkUtilization().

4.25.3.10 uint32 PCM::getNumSockets ( )

Reads number of sockets (CPUs) in the system.

Returns

Number of sockets in the system

Referenced by getAllIncomingQPILinkBytes(), getAllOutgoingQPILinkBytes(), and ServerPCICFGUncore::Server↩ PCICFGUncore().

4.25.3.11 uint32 PCM::getOriginalCPUModel ( ) [inline]

Reads original CPU model id.

Returns

CPU model ID

4.25.3.12 PCIeCounterState PCM::getPCIeCounterState (const uint32 *socket_*)

Get the state of PCIe counter(s)

Parameters

*socket_*	socket of the PCIe controller

Returns

State of PCIe counter(s)

4.25.3.13 uint64 PCM::getQPILinkSpeed ( ) const [inline]

Return QPI Link Speed in GBytes/second.

Warning

Works only for Nehalem-EX (Xeon 7500) and Westmere-EX (Xeon E7) processors

Returns

QPI Link Speed in GBytes/second

Referenced by getIncomingQPILinkUtilization(), getOutgoingQPILinkBytes(), and getOutgoingQPILinkUtilization().

4.25.3.14 uint64 PCM::getQPILinksPerSocket ( ) const [inline]

Returns the number of Intel(r) Quick Path Interconnect(tm) links per socket.

Returns

number of QPI links per socket

References ServerPCICFGUncore::getNumQPIPorts().

Referenced by getAllIncomingQPILinkBytes(), getAllOutgoingQPILinkBytes(), and getSocketIncomingQPILink↩ Bytes().

4.25.3.15 ServerUncorePowerState PCM::getServerUncorePowerState (uint32 *socket*)

Reads the power/energy counter state of a socket (works only on microarchitecture codename SandyBridge-EP)

Parameters

socket	socket id

Returns

State of power counters in the socket

References ServerPCICFGUncore::getDRAMClocks(), ServerPCICFGUncore::getMCCounter(), ServerPCIC↩ FGUncore::getNumMCChannels(), ServerPCICFGUncore::getNumQPIPorts(), ServerPCICFGUncore::getQPI↩ Clocks(), ServerPCICFGUncore::getQPIL0pTxCycles(), and ServerPCICFGUncore::getQPIL1Cycles().

4.25.3.16 bool PCM::getSMT ( )

Checks if SMT (HyperThreading) is enabled.

Returns

true iff SMT (HyperThreading) is enabled.

4.25.3.17 SocketCounterState PCM::getSocketCounterState (uint32 *socket*)

Reads the counter state of a socket.

Parameters

socket	socket id

Returns

State of counters in the socket

Referenced by getSocketCounterState().

4.25.3.18 uint32 PCM::getSocketId (uint32 *core_id*) [inline]

Determines socket of given core.

Parameters

core_id	core identifier

Returns

socket identifier

4.25.3.19 SystemCounterState PCM::getSystemCounterState ( )

Reads the counter state of the system.

System consists of several sockets (CPUs). Socket has a CPU in it. Socket (CPU) consists of several (logical) cores.

Returns

State of counters in the entire system

Referenced by getSystemCounterState().

4.25.3.20 uint32 PCM::getThreadsPerCore ( )

Reads how many hardware threads has a physical core "Hardware thread" is a logical core in a different terminology. If Intel(r) Hyperthreading(tm) is enabled then this function returns 2.

Returns

Number of hardware threads per physical core

Referenced by getCoreIPC(), and getTotalExecUsage().

4.25.3.21 uint64 PCM::getTickCount (uint64 *multiplier* = `1000`, uint32 *core* = `0`)

Return TSC timer value in time units.

Parameters

multiplier	use 1 for seconds, 1000 for ms, 1000000 for mks, etc (default is 1000: ms)

core	core to read on-chip TSC value (default is 0)

Returns

time counter value

References getCoreCounterState(), getInvariantTSC(), and getNominalFrequency().

Referenced by ServerPCICFGUncore::computeQPISpeed().

4.25.3.22 uint64 PCM::getTickCountRDTSCP (uint64 *multiplier =* `1000`)

Return TSC timer value in time units using rdtscp instruction from current core.

Parameters

multiplier	use 1 for seconds, 1000 for ms, 1000000 for mks, etc (default is 1000: ms)

Warning

Processor support is required bit 27 of cpuid EDX must be set, for Windows, Visual Studio 2010 is required

Returns

time counter value

References getNominalFrequency().

4.25.3.23 bool PCM::good ( )

Checks the status of **PCM** (p. 28) object.

Call this method to check if **PCM** (p. 28) gained access to model specific registers. The method is deprecated, see program error code instead.

Returns

true iff access to model specific registers works without problems

4.25.3.24 static bool PCM::initWinRing0Lib ( ) `[static]`

Loads and initializes Winring0 third party library for access to processor model specific and PCI configuration registers.

Returns

returns true in case of success

4.25.3.25 PCM::ErrorCode PCM::program (PCM::ProgramMode *mode_ =* DEFAULT_EVENTS*,* void ∗ *parameter_ =* `NULL`)

Programs performance counters.

Parameters

*mode_*	mode of programming, see ProgramMode definition
*parameter_*	optional parameter for some of programming modes `Call this method before you start using the performance counting routines.`

Warning

> Using this routines with other tools that *program* Performance Monitoring Units (PMUs) on CPUs is not recommended because PMU can not be shared. Tools that are known to program PMUs: Intel(r) VTune(tm), Intel(r) Performance Tuning Utility (PTU). This code may make VTune or PTU measurements invalid. VTune or PTU measurement may make measurement with this code invalid. Please enable either usage of these routines or VTune/PTU/etc.

References ServerPCICFGUncore::computeQPISpeed(), CUSTOM_CORE_EVENTS, EXT_CUSTOM_CORE_↩ EVENTS, and ServerPCICFGUncore::program().

4.25.3.26 void PCM::programPCIeCounters (const PCIeEventCode *event_*)

Program uncore PCIe monitoring event(s)

Parameters

*event_*	a PCIe event to monitor

4.25.3.27 PCM::ErrorCode PCM::programServerUncorePowerMetrics (int *mc_profile,* int *pcu_profile,* int ∗ *freq_bands* = NULL)

Programs uncore power/energy counters on microarchitectures codename SandyBridge-EP and IvyTown.

Parameters

mc_profile	profile for integrated memory controller PMU. See possible profile values in pcm-power.cpp example
pcu_profile	profile for power control unit PMU. See possible profile values in pcm-power.cpp example
freq_bands	array of three integer values for core frequency band monitoring. See usage in pcm-power.↩ cpp example

Call this method before you start using the power counter routines on microarchitecture codename SandyBridge-EP

Warning

> After this call the memory and QPI bandwidth counters on microarchitecture codename SandyBridge-EP will not work.
> Using this routines with other tools that *program* Performance Monitoring Units (PMUs) on CPUs is not recommended because PMU can not be shared. Tools that are known to program PMUs: Intel(r) VTune(tm), Intel(r) Performance Tuning Utility (PTU). This code may make VTune or PTU measurements invalid. VTune or PTU measurement may make measurement with this code invalid. Please enable either usage of these routines or VTune/PTU/etc.

References ServerPCICFGUncore::program_power_metrics().

4.25.3.28 void PCM::resetPMU ( )

Forces PMU reset.

If there is no chance to free up PMU from other applications you might try to call this method at your own risk.

The documentation for this class was generated from the following files:

- **cpucounters.h**
- cpucounters.cpp

4.26 PCM_CPUID_INFO Union Reference

Public Attributes

- int **array** [4]
- struct {
  int **eax**
  int **ebx**
  int **ecx**
  int **edx**
  } **reg**

The documentation for this union was generated from the following file:

- cpucounters.cpp

4.27 ServerPCICFGUncore Class Reference

Object to access uncore counters in a socket/processor with microarchitecture codename SandyBridge-EP (Jaketown) or Ivytown-EP or Ivytown-EX.

```
#include <cpucounters.h>
```

Public Member Functions

- **ServerPCICFGUncore** (uint32 socket_, **PCM** ∗pcm)
  Initialize access data structures.
- void **program** ()
  Program performance counters (disables programming power counters)
- uint64 **getImcReads** ()
  Get the number of integrated controller reads (in cache lines)
- uint64 **getImcWrites** ()
  Get the number of integrated controller writes (in cache lines)
- uint64 **getIncomingDataFlits** (uint32 port)
  Get the number of incoming data flits to the socket through a port.
- uint64 **getOutgoingDataNonDataFlits** (uint32 port)
  Get the number of outgoing data and non-data flits from the socket through a port.
- void **program_power_metrics** (int mc_profile)
  Program power counters (disables programming performance counters)
- uint64 **getQPIClocks** (uint32 port)
  Get number of QPI LL clocks on a QPI port.
- uint64 **getQPIL0pTxCycles** (uint32 port)
  Get number cycles on a QPI port when the link was in a power saving half-lane mode.
- uint64 **getQPIL1Cycles** (uint32 port)
  Get number cycles on a QPI port when the link was in a power saving shutdown mode.
- uint64 **getDRAMClocks** (uint32 channel)
  Get number DRAM channel cycles.

- uint64 **getMCCounter** (uint32 channel, uint32 counter)

  Direct read of memory controller PMU counter (counter meaning depends on the programming: power/performance/etc)
- uint64 **getQPILLCounter** (uint32 port, uint32 counter)

  Direct read of QPI LL PMU counter (counter meaning depends on the programming: power/performance/etc)
- void **freezeCounters** ()

  Freezes event counting.
- void **unfreezeCounters** ()

  Unfreezes event counting.
- uint64 **computeQPISpeed** ()

  Measures/computes the maximum theoretical QPI link bandwidth speed in GByte/seconds.
- void **enableJKTWorkaround** (bool enable)

  Enable correct counting of various LLC events (with memory access perf penalty)
- uint32 **getNumQPIPorts** () const

  Returns the number of detected QPI ports.
- uint32 **getNumMC** () const

  Returns the number of detected integrated memory controllers.
- uint32 **getNumMCChannels** () const

  Returns the total number of detected memory channels on all integrated memory controllers.

4.27.1 Detailed Description

Object to access uncore counters in a socket/processor with microarchitecture codename SandyBridge-EP (Jaketown) or Ivytown-EP or Ivytown-EX.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 ServerPCICFGUncore::ServerPCICFGUncore (uint32 *socket_*, PCM ∗ *pcm*)

Initialize access data structures.

Parameters

*socket_*	socket id
pcm	pointer to **PCM** (p. 28) instance

References PCM::getCPUModel(), and PCM::getNumSockets().

4.27.3 Member Function Documentation

4.27.3.1 uint64 ServerPCICFGUncore::getDRAMClocks (uint32 *channel*)

Get number DRAM channel cycles.

Parameters

channel	channel number

Referenced by PCM::getServerUncorePowerState().

4.27.3.2 uint64 ServerPCICFGUncore::getIncomingDataFlits (uint32 *port*)

Get the number of incoming data flits to the socket through a port.

Parameters

port	QPI port id

4.27.3.3 uint64 ServerPCICFGUncore::getMCCounter (uint32 *channel,* uint32 *counter*)

Direct read of memory controller PMU counter (counter meaning depends on the programming: power/performance/etc)

Parameters

channel	channel number
counter	counter number

Referenced by PCM::getServerUncorePowerState().

4.27.3.4 uint64 ServerPCICFGUncore::getOutgoingDataNonDataFlits (uint32 *port*)

Get the number of outgoing data and non-data flits from the socket through a port.

Parameters

port	QPI port id

References getQPILLCounter().

4.27.3.5 uint64 ServerPCICFGUncore::getQPIClocks (uint32 *port*)

Get number of QPI LL clocks on a QPI port.

Parameters

port	QPI port number

Referenced by computeQPISpeed(), and PCM::getServerUncorePowerState().

4.27.3.6 uint64 ServerPCICFGUncore::getQPIL0pTxCycles (uint32 *port*)

Get number cycles on a QPI port when the link was in a power saving half-lane mode.

Parameters

port	QPI port number

Referenced by PCM::getServerUncorePowerState().

4.27.3.7 uint64 ServerPCICFGUncore::getQPIL1Cycles (uint32 *port*)

Get number cycles on a QPI port when the link was in a power saving shutdown mode.

Parameters

port	QPI port number

Referenced by PCM::getServerUncorePowerState().

4.27.3.8 uint64 ServerPCICFGUncore::getQPILLCounter (uint32 *port,* uint32 *counter*)

Direct read of QPI LL PMU counter (counter meaning depends on the programming: power/performance/etc)

Parameters

port	port number
counter	counter number

Referenced by getOutgoingDataNonDataFlits().

4.27.3.9 void ServerPCICFGUncore::program_power_metrics (int *mc_profile*)

Program power counters (disables programming performance counters)

Parameters

mc_profile	memory controller measurement profile. See description of profiles in pcm-power.cpp

Referenced by PCM::programServerUncorePowerMetrics().

The documentation for this class was generated from the following files:

- **cpucounters.h**
- cpucounters.cpp

4.28 ServerUncorePowerState Class Reference

Server uncore power counter state.

```
#include <cpucounters.h>
```

Public Member Functions

- int32 **getPackageThermalHeadroom** () const
  Returns current thermal headroom below TjMax.

Friends

- class **PCM**
- template<class CounterStateType >
  uint64 **getQPIClocks** (uint32 port, const CounterStateType &before, const CounterStateType &after)
  Returns QPI LL clock ticks.
- template<class CounterStateType >
  uint64 **getQPIL0pTxCycles** (uint32 port, const CounterStateType &before, const CounterStateType &after)
  Returns the number of QPI cycles in power saving half-lane mode.
- template<class CounterStateType >
  uint64 **getQPIL1Cycles** (uint32 port, const CounterStateType &before, const CounterStateType &after)
  Returns the number of QPI cycles in power saving shutdown mode.
- template<class CounterStateType >
  uint64 **getDRAMClocks** (uint32 channel, const CounterStateType &before, const CounterStateType &after)
  Returns DRAM clock ticks.
- template<class CounterStateType >
  uint64 **getMCCounter** (uint32 channel, uint32 counter, const CounterStateType &before, const Counter←StateType &after)
  Direct read of memory controller PMU counter (counter meaning depends on the programming: power/performance/etc)
- template<class CounterStateType >
  uint64 **getPCUCounter** (uint32 counter, const CounterStateType &before, const CounterStateType &after)
  Direct read of power control unit PMU counter (counter meaning depends on the programming: power/performance/etc)

- template<class CounterStateType >
  uint64 **getConsumedEnergy** (const CounterStateType &before, const CounterStateType &after)

  Returns energy consumed by processor, exclusing DRAM (measured in internal units)
- template<class CounterStateType >
  uint64 **getDRAMConsumedEnergy** (const CounterStateType &before, const CounterStateType &after)

  Returns energy consumed by DRAM (measured in internal units)

4.28.1 Detailed Description

Server uncore power counter state.

4.28.2 Friends And Related Function Documentation

4.28.2.1 template<class CounterStateType > uint64 getConsumedEnergy (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Returns energy consumed by processor, exclusing DRAM (measured in internal units)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

4.28.2.2 template<class CounterStateType > uint64 getDRAMClocks (uint32 *channel,* const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Returns DRAM clock ticks.

Parameters

channel	DRAM channel number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

4.28.2.3 template<class CounterStateType > uint64 getDRAMConsumedEnergy (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Returns energy consumed by DRAM (measured in internal units)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

4.28.2.4 template<class CounterStateType > uint64 getMCCounter (uint32 *channel,* uint32 *counter,* const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Direct read of memory controller PMU counter (counter meaning depends on the programming: power/performance/etc)

Parameters

counter	counter number
channel	channel number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

4.28.2.5 template<class CounterStateType > uint64 getPCUCounter (uint32 *counter,* const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Direct read of power control unit PMU counter (counter meaning depends on the programming: power/performance/etc)

Parameters

counter	counter number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

4.28.2.6 template<class CounterStateType > uint64 getQPIClocks (uint32 *port,* const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Returns QPI LL clock ticks.

Parameters

port	QPI port number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

4.28.2.7 template<class CounterStateType > uint64 getQPIL0pTxCycles (uint32 *port,* const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Returns the number of QPI cycles in power saving half-lane mode.

Parameters

port	QPI port number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

4.28.2.8 template<class CounterStateType > uint64 getQPIL1Cycles (uint32 *port,* const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Returns the number of QPI cycles in power saving shutdown mode.

Parameters

port	QPI port number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

The documentation for this class was generated from the following file:

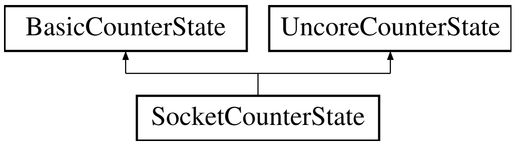
- **cpucounters.h**

4.29 SocketCounterState Class Reference

Socket-wide counter state.

`#include <cpucounters.h>`

Inheritance diagram for SocketCounterState:



Public Member Functions

- void **accumulateCoreState** (const **CoreCounterState** &o)

Protected Member Functions

- void **readAndAggregate** (**MsrHandle** ∗handle)

Friends

- class **PCM**

Additional Inherited Members

4.29.1 Detailed Description

Socket-wide counter state.

The documentation for this class was generated from the following file:

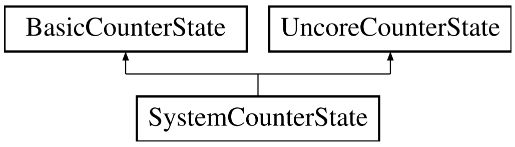
- **cpucounters.h**

4.30 SystemCounterState Class Reference

System-wide counter state.

`#include <cpucounters.h>`

Inheritance diagram for SystemCounterState:



Public Member Functions

- void **accumulateSocketState** (const **SocketCounterState** &o)

Protected Member Functions

- void **readAndAggregate** (**MsrHandle** ∗handle)

Friends

- class **PCM**
- uint64 **getIncomingQPILinkBytes** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get estimation of QPI data traffic per incoming QPI link.
- uint64 **getIncomingQPILinkBytes** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &now)

  Return current value of the counter of QPI data traffic per incoming QPI link.
- double **getOutgoingQPILinkUtilization** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get utilization of outgoing QPI link (0..1)
- uint64 **getOutgoingQPILinkBytes** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get estimation of QPI (data+nondata) traffic per outgoing QPI link.
- uint64 **getOutgoingQPILinkBytes** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &now)

Additional Inherited Members

4.30.1 Detailed Description

System-wide counter state.

4.30.2 Friends And Related Function Documentation

4.30.2.1 uint64 getIncomingQPILinkBytes (uint32 *socketNr,* uint32 *linkNr,* const SystemCounterState & *before,* const SystemCounterState & *after*) `[friend]`

Get estimation of QPI data traffic per incoming QPI link.

Returns an estimation of number of data bytes transferred to a socket over Intel(r) Quick Path Interconnect

Parameters

socketNr	socket identifier
linkNr	linkNr
before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

Number of bytes

4.30.2.2 uint64 getIncomingQPILinkBytes (uint32 *socketNr,* uint32 *linkNr,* const SystemCounterState & *now*) `[friend]`

Return current value of the counter of QPI data traffic per incoming QPI link.

Returns the number of incoming data bytes to a socket over Intel(r) Quick Path Interconnect

Parameters

socketNr	socket identifier
linkNr	linkNr
now	Current System CPU counter state

Returns

Number of bytes

4.30.2.3 uint64 getOutgoingQPILinkBytes (uint32 *socketNr,* uint32 *linkNr,* const **SystemCounterState** & *before,* const **SystemCounterState** & *after*) [friend]

Get estimation of QPI (data+nondata) traffic per outgoing QPI link.

Returns an estimation of number of data bytes transferred from a socket over Intel(r) Quick Path Interconnect

Parameters

socketNr	socket identifier
linkNr	linkNr
before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

Number of bytes

4.30.2.4 double getOutgoingQPILinkUtilization (uint32 *socketNr,* uint32 *linkNr,* const **SystemCounterState** & *before,* const **SystemCounterState** & *after*) [friend]

Get utilization of outgoing QPI link (0..1)

Returns an estimation of utilization of QPI link by (data+nondata) traffic transferred from a socket over Intel(r) Quick Path Interconnect

Parameters

socketNr	socket identifier
linkNr	linkNr
before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

utilization (0..1)

The documentation for this class was generated from the following file:

- **cpucounters.h**

4.31 SystemWideLock Class Reference

The documentation for this class was generated from the following file:

- cpucounters.cpp

4.32 T Struct Reference

Public Member Functions

- **T** (int a)
- bool **operator==** (const **T** &k) const
- **T** (int a)
- bool **operator==** (const **T** &k) const
- **T** (int a)
- bool **operator==** (const **T** &k) const

Public Attributes

- int **key** [1]
- int **data** [3]

The documentation for this struct was generated from the following files:

- memoptest.cpp
- readmem.cpp
- **realtime.cpp**

4.33 TemporalThreadAffinity Class Reference

Public Member Functions

- **TemporalThreadAffinity** (uint32)

The documentation for this class was generated from the following file:

- cpucounters.cpp

4.34 TopologyEntry Struct Reference

Public Attributes

- int32 **os_id**
- int32 **socket**
- int32 **core_id**

The documentation for this struct was generated from the following file:

- **cpucounters.h**

4.35 TSXEvent Struct Reference

Public Attributes

- const char ∗ **name**
- unsigned char **event**

- unsigned char **umask**
- const char ∗ **description**

The documentation for this struct was generated from the following file:

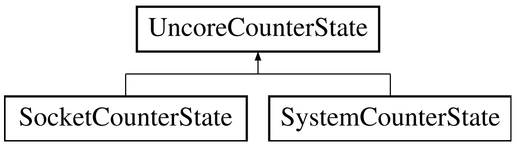
- **pcm-tsx.cpp**

4.36 UncoreCounterState Class Reference

Basic uncore counter state.

`#include <cpucounters.h>`

Inheritance diagram for UncoreCounterState:



Public Member Functions

- **UncoreCounterState** & **operator+=** (const **UncoreCounterState** &o)

Protected Member Functions

- void **readAndAggregate** (**MsrHandle** ∗)

Protected Attributes

- uint64 **UncMCFullWrites**
- uint64 **UncMCNormalReads**
- uint64 **PackageEnergyStatus**
- uint64 **DRAMEnergyStatus**
- uint64 **CStateResidency** [PCM::MAX_C_STATE+1]

Friends

- class **PCM**
- template<class CounterStateType >
  uint64 **getBytesReadFromMC** (const CounterStateType &before, const CounterStateType &after)

  Computes number of bytes read from DRAM memory controllers.
- template<class CounterStateType >
  uint64 **getBytesWrittenToMC** (const CounterStateType &before, const CounterStateType &after)

  Computes number of bytes written to DRAM memory controllers.
- template<class CounterStateType >
  uint64 **getConsumedEnergy** (const CounterStateType &before, const CounterStateType &after)

  Returns energy consumed by processor, exclusing DRAM (measured in internal units)
- template<class CounterStateType >
  uint64 **getDRAMConsumedEnergy** (const CounterStateType &before, const CounterStateType &after)

  Returns energy consumed by DRAM (measured in internal units)

- template<class CounterStateType >
  double **getPackageCStateResidency** (int state, const CounterStateType &before, const CounterStateType &after)

  Computes residency in the package C-state.

4.36.1 Detailed Description

Basic uncore counter state.

Intended only for derivation, but not for the direct use

4.36.2 Friends And Related Function Documentation

4.36.2.1 template<class CounterStateType > uint64 getBytesReadFromMC (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Computes number of bytes read from DRAM memory controllers.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Number of bytes

4.36.2.2 template<class CounterStateType > uint64 getBytesWrittenToMC (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Computes number of bytes written to DRAM memory controllers.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Number of bytes

4.36.2.3 template<class CounterStateType > uint64 getConsumedEnergy (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Returns energy consumed by processor, exclusing DRAM (measured in internal units)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

4.36.2.4 template<class CounterStateType > uint64 getDRAMConsumedEnergy (const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Returns energy consumed by DRAM (measured in internal units)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

4.36.2.5 template<class CounterStateType > double getPackageCStateResidency (int *state,* const CounterStateType & *before,* const CounterStateType & *after*) [friend]

Computes residency in the package C-state.

Parameters

state	C-state
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

residence ratio (0..1): 0 - 0%, 1.0 - 100%

The documentation for this class was generated from the following files:

- **cpucounters.h**
- cpucounters.cpp

4.37 UncoreEventSelectRegister Struct Reference

Public Attributes

- union {
  struct {
  uint64 **event_select**: 8
  uint64 **umask**: 8
  uint64 **reserved1**: 1
  uint64 **occ_ctr_rst**: 1
  uint64 **edge**: 1
  uint64 **reserved2**: 1
  uint64 **enable_pmi**: 1
  uint64 **reserved3**: 1
  uint64 **enable**: 1
  uint64 **invert**: 1
  uint64 **cmask**: 8
  uint64 **reservedx**: 32
  } **fields**
  uint64 **value**
  };

The documentation for this struct was generated from the following file:

- **types.h**

Chapter 5

File Documentation

5.1 client_bw.h File Reference

Interface to access client bandwidth counters.

```
#include "types.h"
#include <unistd.h>
```

Classes

- class **ClientBW**

Macros

- #define **PCM_CLIENT_IMC_BAR_OFFSET** (0x0048)
- #define **PCM_CLIENT_IMC_DRAM_DATA_READS** (0x5050)
- #define **PCM_CLIENT_IMC_DRAM_DATA_WRITES** (0x5054)
- #define **PCM_CLIENT_IMC_MMAP_SIZE** (0x6000)

5.1.1 Detailed Description

Interface to access client bandwidth counters.

5.2 cpuasynchcounter.h File Reference

Implementation of a POSIX thread that periodically saves the current state of counters and exposes them to other threads.

```
#include <pthread.h>
#include <stdlib.h>
#include "cpucounters.h"
```

Classes

- class **AsynchronCounterState**

Macros

- #define **DELAY** 1

Functions

- void ∗ **UpdateCounters** (void ∗)

5.2.1 Detailed Description

Implementation of a POSIX thread that periodically saves the current state of counters and exposes them to other threads.

5.3 cpucounters.h File Reference

Main CPU counters header.

```
#include "types.h"
#include "msr.h"
#include "pci.h"
#include "client_bw.h"
#include "width_extender.h"
#include <vector>
#include <limits>
#include <string.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

Classes

- struct **TopologyEntry**
- class **ServerPCICFGUncore**

  Object to access uncore counters in a socket/processor with microarchitecture codename SandyBridge-EP (Jaketown) or Ivytown-EP or Ivytown-EX.
- class **PCIeCounterState**
- class **PCM**

  CPU Performance Monitor.
- struct **PCM::CustomCoreEventDescription**

  Custom Core event description.
- struct **PCM::ExtendedCustomCoreEventDescription**

  Extended custom core event description.
- class **BasicCounterState**

  Basic core counter state.
- class **ServerUncorePowerState**

  Server uncore power counter state.
- class **UncoreCounterState**

  Basic uncore counter state.
- class **CoreCounterState**

(Logical) core-wide counter state
- class **SocketCounterState**

  Socket-wide counter state.
- class **SystemCounterState**

  System-wide counter state.

Macros

- #define **INTEL_PCM_VERSION** "V2.6 (2013-11-04 13:43:31 +0100 ID=db05e43)"
- #define **INTELPCM_API**

Functions

- template<class CounterStateType >
  uint64 **getQPIClocks** (uint32 port, const CounterStateType &before, const CounterStateType &after)

  Returns QPI LL clock ticks.
- template<class CounterStateType >
  int32 **getThermalHeadroom** (const CounterStateType &, const CounterStateType &after)
- template<class CounterStateType >
  uint64 **getQPIL0pTxCycles** (uint32 port, const CounterStateType &before, const CounterStateType &after)

  Returns the number of QPI cycles in power saving half-lane mode.
- template<class CounterStateType >
  uint64 **getQPIL1Cycles** (uint32 port, const CounterStateType &before, const CounterStateType &after)

  Returns the number of QPI cycles in power saving shutdown mode.
- template<class CounterStateType >
  double **getNormalizedQPIL0pTxCycles** (uint32 port, const CounterStateType &before, const Counter↩ StateType &after)

  Returns the ratio of QPI cycles in power saving half-lane mode.
- template<class CounterStateType >
  double **getNormalizedQPIL1Cycles** (uint32 port, const CounterStateType &before, const CounterStateType &after)

  Returns the ratio of QPI cycles in power saving shutdown mode.
- template<class CounterStateType >
  uint64 **getDRAMClocks** (uint32 channel, const CounterStateType &before, const CounterStateType &after)

  Returns DRAM clock ticks.
- template<class CounterStateType >
  uint64 **getMCCounter** (uint32 channel, uint32 counter, const CounterStateType &before, const Counter↩ StateType &after)

  Direct read of memory controller PMU counter (counter meaning depends on the programming: power/performance/etc)
- template<class CounterStateType >
  uint64 **getPCUCounter** (uint32 counter, const CounterStateType &before, const CounterStateType &after)

  Direct read of power control unit PMU counter (counter meaning depends on the programming: power/performance/etc)
- template<class CounterStateType >
  uint64 **getPCUClocks** (const CounterStateType &before, const CounterStateType &after)

  Returns clock ticks of power control unit.
- template<class CounterStateType >
  uint64 **getConsumedEnergy** (const CounterStateType &before, const CounterStateType &after)

  Returns energy consumed by processor, exclusing DRAM (measured in internal units)
- template<class CounterStateType >
  uint64 **getDRAMConsumedEnergy** (const CounterStateType &before, const CounterStateType &after)

  Returns energy consumed by DRAM (measured in internal units)
- template<class CounterStateType >
  double **getConsumedJoules** (const CounterStateType &before, const CounterStateType &after)

Returns Joules consumed by processor (excluding DRAM)

- template<class CounterStateType >
  double **getDRAMConsumedJoules** (const CounterStateType &before, const CounterStateType &after)

  Returns Joules consumed by DRAM.
- INTELPCM_API **SystemCounterState getSystemCounterState** ()

  Reads the counter state of the system.
- INTELPCM_API **SocketCounterState getSocketCounterState** (uint32 socket)

  Reads the counter state of a socket.
- INTELPCM_API **CoreCounterState getCoreCounterState** (uint32 core)

  Reads the counter state of a (logical) core.
- template<class CounterStateType >
  double **getIPC** (const CounterStateType &before, const CounterStateType &after)

  Computes average number of retired instructions per core cycle (IPC)
- template<class CounterStateType >
  uint64 **getInstructionsRetired** (const CounterStateType &before, const CounterStateType &after)

  Computes the number of retired instructions.
- template<class CounterStateType >
  double **getExecUsage** (const CounterStateType &before, const CounterStateType &after)

  Computes average number of retired instructions per time intervall.
- template<class CounterStateType >
  uint64 **getInstructionsRetired** (const CounterStateType &now)

  Computes the number of retired instructions.
- template<class CounterStateType >
  uint64 **getCycles** (const CounterStateType &before, const CounterStateType &after)

  Computes the number core clock cycles when signal on a specific core is running (not halted)
- template<class CounterStateType >
  uint64 **getRefCycles** (const CounterStateType &before, const CounterStateType &after)

  Computes the number of reference clock cycles while clock signal on the core is running.
- template<class CounterStateType >
  uint64 **getCycles** (const CounterStateType &now)

  Computes the number executed core clock cycles.
- double **getCoreIPC** (const **SystemCounterState** &before, const **SystemCounterState** &after)

  Computes average number of retired instructions per core cycle for the entire system combining instruction counts from logical cores to corresponding physical cores.
- double **getTotalExecUsage** (const **SystemCounterState** &before, const **SystemCounterState** &after)

  Computes average number of retired instructions per time intervall for the entire system combining instruction counts from logical cores to corresponding physical cores.
- template<class CounterStateType >
  double **getAverageFrequency** (const CounterStateType &before, const CounterStateType &after)

  Computes average core frequency also taking Intel Turbo Boost technology into account.
- template<class CounterStateType >
  double **getActiveAverageFrequency** (const CounterStateType &before, const CounterStateType &after)

  Computes average core frequency when not in powersaving C0-state (also taking Intel Turbo Boost technology into account)
- template<class CounterStateType >
  double **getRelativeFrequency** (const CounterStateType &before, const CounterStateType &after)

  Computes average core frequency also taking Intel Turbo Boost technology into account.
- template<class CounterStateType >
  double **getActiveRelativeFrequency** (const CounterStateType &before, const CounterStateType &after)

  Computes average core frequency when not in powersaving C0-state (also taking Intel Turbo Boost technology into account)
- template<class CounterStateType >
  double **getCyclesLostDueL3CacheMisses** (const CounterStateType &before, const CounterStateType &after)

Estimates how many core cycles were potentially lost due to L3 cache misses.

- template<class CounterStateType >
double **getCyclesLostDueL2CacheMisses** (const CounterStateType &before, const CounterStateType &after)

  Estimates how many core cycles were potentially lost due to missing L2 cache but still hitting L3 cache.
- template<class CounterStateType >
double **getL2CacheHitRatio** (const CounterStateType &before, const CounterStateType &after)

  Computes L2 cache hit ratio.
- template<class CounterStateType >
double **getL3CacheHitRatio** (const CounterStateType &before, const CounterStateType &after)

  Computes L3 cache hit ratio.
- template<class CounterStateType >
uint64 **getL3CacheMisses** (const CounterStateType &before, const CounterStateType &after)

  Computes number of L3 cache misses.
- template<class CounterStateType >
uint64 **getL2CacheMisses** (const CounterStateType &before, const CounterStateType &after)

  Computes number of L2 cache misses.
- template<class CounterStateType >
uint64 **getL2CacheHits** (const CounterStateType &before, const CounterStateType &after)

  Computes number of L2 cache hits.
- template<class CounterStateType >
uint64 **getL3CacheHitsNoSnoop** (const CounterStateType &before, const CounterStateType &after)

  Computes number of L3 cache hits where no snooping in sibling L2 caches had to be done.
- template<class CounterStateType >
uint64 **getL3CacheHitsSnoop** (const CounterStateType &before, const CounterStateType &after)

  Computes number of L3 cache hits where snooping in sibling L2 caches had to be done.
- template<class CounterStateType >
uint64 **getL3CacheHits** (const CounterStateType &before, const CounterStateType &after)

  Computes total number of L3 cache hits.
- template<class CounterStateType >
uint64 **getInvariantTSC** (const CounterStateType &before, const CounterStateType &after)

  Computes number of invariant time stamp counter ticks.
- template<class CounterStateType >
double **getCoreCStateResidency** (int state, const CounterStateType &before, const CounterStateType &after)

  Computes residency in the core C-state.
- template<class CounterStateType >
double **getPackageCStateResidency** (int state, const CounterStateType &before, const CounterStateType &after)

  Computes residency in the package C-state.
- template<class CounterStateType >
uint64 **getBytesReadFromMC** (const CounterStateType &before, const CounterStateType &after)

  Computes number of bytes read from DRAM memory controllers.
- template<class CounterStateType >
uint64 **getBytesWrittenToMC** (const CounterStateType &before, const CounterStateType &after)

  Computes number of bytes written to DRAM memory controllers.
- template<class CounterStateType >
uint64 **getNumberOfCustomEvents** (int32 eventCounterNr, const CounterStateType &before, const CounterStateType &after)

  Returns the number of occured custom core events.
- uint64 **getIncomingQPILinkBytes** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get estimation of QPI data traffic per incoming QPI link.

- double **getIncomingQPILinkUtilization** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get data utilization of incoming QPI link (0..1)
- double **getOutgoingQPILinkUtilization** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get utilization of outgoing QPI link (0..1)
- uint64 **getOutgoingQPILinkBytes** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get estimation of QPI (data+nondata) traffic per outgoing QPI link.
- uint64 **getAllIncomingQPILinkBytes** (const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get estimation of total QPI data traffic.
- uint64 **getAllOutgoingQPILinkBytes** (const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get estimation of total QPI data+nondata traffic.
- uint64 **getIncomingQPILinkBytes** (uint32 socketNr, uint32 linkNr, const **SystemCounterState** &now)

  Return current value of the counter of QPI data traffic per incoming QPI link.
- uint64 **getSocketIncomingQPILinkBytes** (uint32 socketNr, const **SystemCounterState** &now)

  Get estimation of total QPI data traffic for this socket.
- uint64 **getAllIncomingQPILinkBytes** (const **SystemCounterState** &now)

  Get estimation of Socket QPI data traffic.
- double **getQPItoMCTrafficRatio** (const **SystemCounterState** &before, const **SystemCounterState** &after)

  Get QPI data to Memory Controller traffic ratio.
- uint64 **getNumberOfEvents** (**PCIeCounterState** before, **PCIeCounterState** after)

  Returns the raw count of PCIe events.

5.3.1 Detailed Description

Main CPU counters header.

Include this header file if you want to access CPU counters (core and uncore - including memory controller chips and QPI)

5.3.2 Function Documentation

5.3.2.1 template<class CounterStateType > double getActiveAverageFrequency (const CounterStateType & *before,* const CounterStateType & *after*)

Computes average core frequency when not in powersaving C0-state (also taking Intel Turbo Boost technology into account)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

frequency in Hz

References PCM::getInstance(), and PCM::getNominalFrequency().

5.3.2.2 template<class CounterStateType > double getActiveRelativeFrequency (const CounterStateType & *before,* const CounterStateType & *after*)

Computes average core frequency when not in powersaving C0-state (also taking Intel Turbo Boost technology into account)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Fraction of nominal frequency (if $>$1.0 then Turbo was working during the measurement)

5.3.2.3 uint64 getAllIncomingQPILinkBytes (const SystemCounterState & *before*, const SystemCounterState & *after*) [inline]

Get estimation of total QPI data traffic.

Returns an estimation of number of data bytes transferred to all sockets over all Intel(r) Quick Path Interconnect links

Parameters

before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

Number of bytes

References getIncomingQPILinkBytes(), PCM::getInstance(), PCM::getNumSockets(), and PCM::getQPILinksPer↩ Socket().

Referenced by getQPItoMCTrafficRatio().

5.3.2.4 uint64 getAllIncomingQPILinkBytes (const SystemCounterState & *now*) [inline]

Get estimation of Socket QPI data traffic.

Returns an estimation of number of data bytes transferred to all sockets over all Intel(r) Quick Path Interconnect links

Parameters

now	System CPU counter state

Returns

Number of bytes

References PCM::getInstance(), PCM::getNumSockets(), and getSocketIncomingQPILinkBytes().

5.3.2.5 uint64 getAllOutgoingQPILinkBytes (const SystemCounterState & *before*, const SystemCounterState & *after*) [inline]

Get estimation of total QPI data+nondata traffic.

Returns an estimation of number of data and non-data bytes transferred from all sockets over all Intel(r) Quick Path Interconnect links

Parameters

before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

Number of bytes

References PCM::getInstance(), PCM::getNumSockets(), getOutgoingQPILinkBytes(), and PCM::getQPILinksPer↩
Socket().

5.3.2.6 template<class CounterStateType > double getAverageFrequency (const CounterStateType & *before,* const CounterStateType & *after*)

Computes average core frequency also taking Intel Turbo Boost technology into account.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

frequency in Hz

References PCM::getInstance(), and PCM::getNominalFrequency().

5.3.2.7 template<class CounterStateType > uint64 getBytesReadFromMC (const CounterStateType & *before,* const CounterStateType & *after*)

Computes number of bytes read from DRAM memory controllers.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Number of bytes

Referenced by getQPItoMCTrafficRatio().

5.3.2.8 template<class CounterStateType > uint64 getBytesWrittenToMC (const CounterStateType & *before,* const CounterStateType & *after*)

Computes number of bytes written to DRAM memory controllers.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Number of bytes

Referenced by getQPItoMCTrafficRatio().

5.3.2.9 template< class CounterStateType > uint64 getConsumedEnergy (const CounterStateType & *before,* const CounterStateType & *after*)

Returns energy consumed by processor, exclusing DRAM (measured in internal units)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Referenced by getConsumedJoules().

5.3.2.10 template<class CounterStateType > double getConsumedJoules (const CounterStateType & *before*, const CounterStateType & *after*)

Returns Joules consumed by processor (excluding DRAM)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

References getConsumedEnergy(), PCM::getInstance(), and PCM::getJoulesPerEnergyUnit().

5.3.2.11 INTELPCM_API CoreCounterState getCoreCounterState (uint32 *core*)

Reads the counter state of a (logical) core.

Helper function. Uses **PCM** (p. 28) object to access counters.

Parameters

core	core id

Returns

State of counters in the core

References PCM::getCoreCounterState(), and PCM::getInstance().

5.3.2.12 template<class CounterStateType > double getCoreCStateResidency (int *state*, const CounterStateType & *before*, const CounterStateType & *after*) `[inline]`

Computes residency in the core C-state.

Parameters

state	C-state
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

residence ratio (0..1): 0 - 0%, 1.0 - 100%

References PCM::getInstance(), getInvariantTSC(), getRefCycles(), and PCM::isCoreCStateResidency↩ Supported().

5.3.2.13 double getCoreIPC (const SystemCounterState & *before*, const SystemCounterState & *after*) `[inline]`

Computes average number of retired instructions per core cycle for the entire system combining instruction counts from logical cores to corresponding physical cores.

Use this metric to evaluate IPC improvement between SMT(Hyperthreading) on and SMT off.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

IPC

References PCM::getInstance(), getIPC(), and PCM::getThreadsPerCore().

5.3.2.14 template<class CounterStateType > uint64 getCycles (const CounterStateType & *before*, const CounterStateType & *after*)

Computes the number core clock cycles when signal on a specific core is running (not halted)

Returns number of used cycles (halted cyles are not counted). The counter does not advance in the following conditions:

- an ACPI C-state is other than C0 for normal operation
- HLT
- STPCLK+ pin is asserted
- being throttled by TM1
- during the frequency switching phase of a performance state transition

The performance counter for this event counts across performance state transitions using different core clock frequencies

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

number core clock cycles

5.3.2.15 template<class CounterStateType > uint64 getCycles (const CounterStateType & *now*)

Computes the number executed core clock cycles.

Returns number of used cycles (halted cyles are not counted).

Parameters

now	Current CPU counter state

Returns

number core clock cycles

5.3.2.16 template<class CounterStateType > double getCyclesLostDueL2CacheMisses (const CounterStateType & *before*, const CounterStateType & *after*)

Estimates how many core cycles were potentially lost due to missing L2 cache but still hitting L3 cache.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)
Currently not supported on Intel(R) Atom(tm) processor

Returns

ratio that is usually beetween 0 and 1 ; in some cases could be >1.0 due to a lower access latency estimation

References PCM::getCPUModel(), and PCM::getInstance().

5.3.2.17 template<class CounterStateType > double getCyclesLostDueL3CacheMisses (const CounterStateType & *before,* const CounterStateType & *after*)

Estimates how many core cycles were potentially lost due to L3 cache misses.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

ratio that is usually beetween 0 and 1 ; in some cases could be >1.0 due to a lower memory latency estimation

References PCM::getCPUModel(), and PCM::getInstance().

5.3.2.18 template<class CounterStateType > uint64 getDRAMClocks (uint32 *channel,* const CounterStateType & *before,* const CounterStateType & *after*)

Returns DRAM clock ticks.

Parameters

channel	DRAM channel number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

5.3.2.19 template<class CounterStateType > uint64 getDRAMConsumedEnergy (const CounterStateType & *before,* const CounterStateType & *after*)

Returns energy consumed by DRAM (measured in internal units)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Referenced by getDRAMConsumedJoules().

5.3.2.20 template< class CounterStateType > double getDRAMConsumedJoules (const CounterStateType & *before,* const CounterStateType & *after*)

Returns Joules consumed by DRAM.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

References getDRAMConsumedEnergy(), PCM::getInstance(), and PCM::getJoulesPerEnergyUnit().

5.3.2.21 template< class CounterStateType > double getExecUsage (const CounterStateType & *before,* const CounterStateType & *after*)

Computes average number of retired instructions per time intervall.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

usage

Referenced by getTotalExecUsage().

5.3.2.22 uint64 getIncomingQPILinkBytes (uint32 *socketNr,* uint32 *linkNr,* const SystemCounterState & *before,* const SystemCounterState & *after*) [inline]

Get estimation of QPI data traffic per incoming QPI link.

Returns an estimation of number of data bytes transferred to a socket over Intel(r) Quick Path Interconnect

Parameters

socketNr	socket identifier
linkNr	linkNr
before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

Number of bytes

Referenced by getAllIncomingQPILinkBytes(), getIncomingQPILinkUtilization(), and getSocketIncomingQPILink↩
Bytes().

5.3.2.23 uint64 getIncomingQPILinkBytes (uint32 *socketNr,* uint32 *linkNr,* const SystemCounterState & *now*) [inline]

Return current value of the counter of QPI data traffic per incoming QPI link.

Returns the number of incoming data bytes to a socket over Intel(r) Quick Path Interconnect

Parameters

socketNr	socket identifier
linkNr	linkNr
now	Current System CPU counter state

Returns

Number of bytes

5.3.2.24 double getIncomingQPILinkUtilization (uint32 *socketNr,* uint32 *linkNr,* const SystemCounterState & *before,* const SystemCounterState & *after*) [inline]

Get data utilization of incoming QPI link (0..1)

Returns an estimation of utilization of QPI link by data traffic transferred to a socket over Intel(r) Quick Path Interconnect

Parameters

socketNr	socket identifier
linkNr	linkNr
before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

utilization (0..1)

References getIncomingQPILinkBytes(), PCM::getInstance(), getInvariantTSC(), PCM::getNominalFrequency(), PCM::getNumCores(), and PCM::getQPILinkSpeed().

5.3.2.25 template<class CounterStateType > uint64 getInstructionsRetired (const CounterStateType & *before,* const CounterStateType & *after*)

Computes the number of retired instructions.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

number of retired instructions

5.3.2.26 template<class CounterStateType > uint64 getInstructionsRetired (const CounterStateType & *now*)

Computes the number of retired instructions.

Parameters

now	Current CPU counter state

Returns

number of retired instructions

5.3.2.27 template< class CounterStateType > uint64 getInvariantTSC (const CounterStateType & *before,* const CounterStateType & *after*)

Computes number of invariant time stamp counter ticks.

This counter counts irrespectively of C-, P- or T-states

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

number of time stamp counter ticks

Referenced by getCoreCStateResidency(), getIncomingQPILinkUtilization(), getOutgoingQPILinkBytes(), get↩ OutgoingQPILinkUtilization(), getPackageCStateResidency(), and PCM::getTickCount().

5.3.2.28 template< class CounterStateType > double getIPC (const CounterStateType & *before,* const CounterStateType & *after*)

Computes average number of retired instructions per core cycle (IPC)

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

IPC

Referenced by getCoreIPC().

5.3.2.29 template< class CounterStateType > double getL2CacheHitRatio (const CounterStateType & *before,* const CounterStateType & *after*)

Computes L2 cache hit ratio.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

value between 0 and 1

References PCM::getCPUModel(), and PCM::getInstance().

5.3.2.30 template< class CounterStateType > uint64 getL2CacheHits (const CounterStateType & *before,* const CounterStateType & *after*)

Computes number of L2 cache hits.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of hits

References PCM::getCPUModel(), and PCM::getInstance().

5.3.2.31 template<class CounterStateType > uint64 getL2CacheMisses (const CounterStateType & *before,* const CounterStateType & *after*)

Computes number of L2 cache misses.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of misses

References PCM::getCPUModel(), and PCM::getInstance().

5.3.2.32 template<class CounterStateType > double getL3CacheHitRatio (const CounterStateType & *before,* const CounterStateType & *after*)

Computes L3 cache hit ratio.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

value between 0 and 1

References PCM::getCPUModel(), and PCM::getInstance().

5.3.2.33 template<class CounterStateType > uint64 getL3CacheHits (const CounterStateType & *before,* const CounterStateType & *after*)

Computes total number of L3 cache hits.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of hits

References PCM::getCPUModel(), PCM::getInstance(), getL3CacheHitsNoSnoop(), and getL3CacheHitsSnoop().

5.3.2.34 template< class CounterStateType > uint64 getL3CacheHitsNoSnoop (const CounterStateType & *before,* const CounterStateType & *after*)

Computes number of L3 cache hits where no snooping in sibling L2 caches had to be done.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of hits

References PCM::getCPUModel(), and PCM::getInstance().

Referenced by getL3CacheHits().

5.3.2.35 template< class CounterStateType > uint64 getL3CacheHitsSnoop (const CounterStateType & *before,* const CounterStateType & *after*)

Computes number of L3 cache hits where snooping in sibling L2 caches had to be done.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of hits

References PCM::getCPUModel(), and PCM::getInstance().

Referenced by getL3CacheHits().

5.3.2.36 template< class CounterStateType > uint64 getL3CacheMisses (const CounterStateType & *before,* const CounterStateType & *after*)

Computes number of L3 cache misses.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Warning

Works only in the DEFAULT_EVENTS programming mode (see program() method)

Returns

number of misses

References PCM::getCPUModel(), and PCM::getInstance().

5.3.2.37 template<class CounterStateType > uint64 getMCCounter (uint32 *channel,* uint32 *counter,* const CounterStateType & *before,* const CounterStateType & *after*)

Direct read of memory controller PMU counter (counter meaning depends on the programming: power/performance/etc)

Parameters

counter	counter number
channel	channel number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

5.3.2.38 template<class CounterStateType > double getNormalizedQPIL0pTxCycles (uint32 *port,* const CounterStateType & *before,* const CounterStateType & *after*)

Returns the ratio of QPI cycles in power saving half-lane mode.

Parameters

port	QPI port number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

0..1 - ratio of QPI cycles in power saving half-lane mode

References getQPIClocks(), and getQPIL0pTxCycles().

5.3.2.39 template<class CounterStateType > double getNormalizedQPIL1Cycles (uint32 *port,* const CounterStateType & *before,* const CounterStateType & *after*)

Returns the ratio of QPI cycles in power saving shutdown mode.

Parameters

port	QPI port number
before	CPU counter state before the experiment

after	CPU counter state after the experiment

Returns

0..1 - ratio of QPI cycles in power saving shutdown mode

References getQPIClocks(), and getQPIL1Cycles().

5.3.2.40 template<class CounterStateType > uint64 getNumberOfCustomEvents (int32 *eventCounterNr,* const CounterStateType & *before,* const CounterStateType & *after*)

Returns the number of occured custom core events.

Read number of events programmed with the `CUSTOM_CORE_EVENTS`

Parameters

eventCounterNr	Event/counter number (value from 0 to 3)
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Number of bytes

5.3.2.41 uint64 getNumberOfEvents (PCIeCounterState *before,* PCIeCounterState *after*) [inline]

Returns the raw count of PCIe events.

Parameters

before	PCIe counter state before the experiment
after	PCIe counter state after the experiment

5.3.2.42 uint64 getOutgoingQPILinkBytes (uint32 *socketNr,* uint32 *linkNr,* const SystemCounterState & *before,* const SystemCounterState & *after*) [inline]

Get estimation of QPI (data+nondata) traffic per outgoing QPI link.

Returns an estimation of number of data bytes transferred from a socket over Intel(r) Quick Path Interconnect

Parameters

socketNr	socket identifier
linkNr	linkNr
before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

Number of bytes

References PCM::getInstance(), getInvariantTSC(), PCM::getNominalFrequency(), PCM::getNumCores(), get↩OutgoingQPILinkUtilization(), and PCM::getQPILinkSpeed().

Referenced by getAllOutgoingQPILinkBytes().

5.3.2.43 double getOutgoingQPILinkUtilization (uint32 *socketNr,* uint32 *linkNr,* const SystemCounterState & *before,* const SystemCounterState & *after*) [inline]

Get utilization of outgoing QPI link (0..1)

Returns an estimation of utilization of QPI link by (data+nondata) traffic transferred from a socket over Intel(r) Quick Path Interconnect

Parameters

socketNr	socket identifier
linkNr	linkNr
before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

utilization (0..1)

References PCM::getInstance(), getInvariantTSC(), PCM::getNominalFrequency(), PCM::getNumCores(), and P↩
CM::getQPILinkSpeed().

Referenced by getOutgoingQPILinkBytes().

5.3.2.44 template<class CounterStateType > double getPackageCStateResidency (int *state,* const CounterStateType & *before,* const CounterStateType & *after*) [inline]

Computes residency in the package C-state.

Parameters

state	C-state
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

residence ratio (0..1): 0 - 0%, 1.0 - 100%

References getInvariantTSC().

5.3.2.45 template<class CounterStateType > uint64 getPCUClocks (const CounterStateType & *before,* const CounterStateType & *after*)

Returns clock ticks of power control unit.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

References getPCUCounter().

5.3.2.46 template<class CounterStateType > uint64 getPCUCounter (uint32 *counter,* const CounterStateType & *before,* const CounterStateType & *after*)

Direct read of power control unit PMU counter (counter meaning depends on the programming: power/performance/etc)

Parameters

counter	counter number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Referenced by getPCUClocks().

5.3.2.47 template<class CounterStateType > uint64 getQPIClocks (uint32 *port*, const CounterStateType & *before*, const CounterStateType & *after*)

Returns QPI LL clock ticks.

Parameters

port	QPI port number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Referenced by getNormalizedQPIL0pTxCycles(), and getNormalizedQPIL1Cycles().

5.3.2.48 template<class CounterStateType > uint64 getQPIL0pTxCycles (uint32 *port*, const CounterStateType & *before*, const CounterStateType & *after*)

Returns the number of QPI cycles in power saving half-lane mode.

Parameters

port	QPI port number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Referenced by getNormalizedQPIL0pTxCycles().

5.3.2.49 template<class CounterStateType > uint64 getQPIL1Cycles (uint32 *port*, const CounterStateType & *before*, const CounterStateType & *after*)

Returns the number of QPI cycles in power saving shutdown mode.

Parameters

port	QPI port number
before	CPU counter state before the experiment
after	CPU counter state after the experiment

Referenced by getNormalizedQPIL1Cycles().

5.3.2.50 double getQPItoMCTrafficRatio (const SystemCounterState & *before*, const SystemCounterState & *after*) [inline]

Get QPI data to Memory Controller traffic ratio.

Ideally for NUMA-optmized programs the ratio should be close to 0.

Parameters

before	System CPU counter state before the experiment

after	System CPU counter state after the experiment

Returns

Ratio

References getAllIncomingQPILinkBytes(), getBytesReadFromMC(), and getBytesWrittenToMC().

5.3.2.51 template<class CounterStateType > uint64 getRefCycles (const CounterStateType & *before,* const CounterStateType & *after*)

Computes the number of reference clock cycles while clock signal on the core is running.

The reference clock operates at a fixed frequency, irrespective of core frequency changes due to performance state transitions. See Intel(r) Software Developer's Manual for more details

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

number core clock cycles

Referenced by getCoreCStateResidency().

5.3.2.52 template<class CounterStateType > double getRelativeFrequency (const CounterStateType & *before,* const CounterStateType & *after*)

Computes average core frequency also taking Intel Turbo Boost technology into account.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

Fraction of nominal frequency

5.3.2.53 INTELPCM_API SocketCounterState getSocketCounterState (uint32 *socket*)

Reads the counter state of a socket.

Helper function. Uses **PCM** (p. 28) object to access counters.

Parameters

socket	socket id

Returns

State of counters in the socket

References PCM::getInstance(), and PCM::getSocketCounterState().

5.3.2.54 uint64 getSocketIncomingQPILinkBytes (uint32 *socketNr*, const **SystemCounterState** & *now*) [inline]

Get estimation of total QPI data traffic for this socket.

Returns an estimation of number of bytes transferred to this sockets over all Intel(r) Quick Path Interconnect links on this socket

Parameters

before	System CPU counter state before the experiment
after	System CPU counter state after the experiment

Returns

Number of bytes

References getIncomingQPILinkBytes(), PCM::getInstance(), and PCM::getQPILinksPerSocket().

Referenced by getAllIncomingQPILinkBytes().

5.3.2.55 INTELPCM_API SystemCounterState getSystemCounterState ( )

Reads the counter state of the system.

Helper function. Uses **PCM** (p. 28) object to access counters.

System consists of several sockets (CPUs). Socket has a CPU in it. Socket (CPU) consists of several (logical) cores.

Returns

State of counters in the entire system

References PCM::getInstance(), and PCM::getSystemCounterState().

5.3.2.56 double getTotalExecUsage (const **SystemCounterState** & *before*, const **SystemCounterState** & *after*) [inline]

Computes average number of retired instructions per time intervall for the entire system combining instruction counts from logical cores to corresponding physical cores.

Use this metric to evaluate cores utilization improvement between SMT(Hyperthreading) on and SMT off.

Parameters

before	CPU counter state before the experiment
after	CPU counter state after the experiment

Returns

usage

References getExecUsage(), PCM::getInstance(), and PCM::getThreadsPerCore().

5.4 cpucounterstest.cpp File Reference

Example of using CPU counters: implements a simple performance counter monitoring utility.

```
#include <iostream>
#include <unistd.h>
#include <signal.h>
#include <math.h>
#include <iomanip>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <assert.h>
#include "cpucounters.h"
#include "utils.h"
```

Macros

- #define **HACK_TO_REMOVE_DUPLICATE_ERROR**
- #define **SIZE** (10000000)
- #define **DELAY** 1

Functions

- template< class IntType >
  double **float_format** (IntType n)
- std::string **temp_format** (int32 t)
- void **print_help** (char ∗prog_name)
- void **print_output** (**PCM** ∗m, const std::vector< **CoreCounterState** > &cstates1, const std::vector< **Core↩ CounterState** > &cstates2, const std::vector< **SocketCounterState** > &sktstate1, const std::vector< **SocketCounterState** > &sktstate2, const **SystemCounterState** &sstate1, const **SystemCounterState** &sstate2, const int cpu_model, const bool show_core_output, const bool show_socket_output, const bool show_system_output)
- void **print_csv_header** (**PCM** ∗m, const int cpu_model, const bool show_core_output, const bool show_↩ socket_output, const bool show_system_output)
- void **print_csv** (**PCM** ∗m, const std::vector< **CoreCounterState** > &cstates1, const std::vector< **Core↩ CounterState** > &cstates2, const std::vector< **SocketCounterState** > &sktstate1, const std::vector< **SocketCounterState** > &sktstate2, const **SystemCounterState** &sstate1, const **SystemCounterState** &sstate2, const int cpu_model, const bool show_core_output, const bool show_socket_output, const bool show_system_output)
- int **main** (int argc, char ∗argv[ ])

5.4.1 Detailed Description

Example of using CPU counters: implements a simple performance counter monitoring utility.

5.5 msr.h File Reference

Low level interface to access hardware model specific registers.

```
#include "types.h"
```

Classes

- class **MsrHandle**

5.5.1 Detailed Description

Low level interface to access hardware model specific registers.

Implemented and tested for Linux and 64-bit Windows 7

5.6 pci.h File Reference

Low level interface to access PCI configuration space.

```
#include "types.h"
#include <unistd.h>
```

Classes

- class **PciHandle**
- class **PciHandleM**
- class **PciHandleMM**

Macros

- #define **PCM_USE_PCI_MM_LINUX**
- #define **PciHandleM PciHandleMM**

5.6.1 Detailed Description

Low level interface to access PCI configuration space.

5.7 pcm-memory.cpp File Reference

Example of using CPU counters: implements a performance counter monitoring utility for memory controller channels.

```
#include <iostream>
#include <unistd.h>
#include <signal.h>
#include <math.h>
#include <iomanip>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <assert.h>
#include "cpucounters.h"
#include "utils.h"
```

Macros

- #define **HACK_TO_REMOVE_DUPLICATE_ERROR**
- #define **READ** 0
- #define **WRITE** 1
- #define **PARTIAL** 2

Functions

- void **print_help** (char ∗prog_name)
- void **display_bandwidth** (float ∗iMC_Rd_socket_chan, float ∗iMC_Wr_socket_chan, float ∗iMC_Rd_socket, float ∗iMC_Wr_socket, uint32 numSockets, uint32 num_imc_channels, uint64 ∗partial_write)
- void **calculate_bandwidth** (**PCM** ∗m, const **ServerUncorePowerState** uncState1[ ], const **ServerUncore↩ PowerState** uncState2[ ], uint64 elapsedTime)
- int **main** (int argc, char ∗argv[ ])

Variables

- const uint32 **max_sockets** = 4
- const uint32 **max_imc_channels** = 8

5.7.1 Detailed Description

Example of using CPU counters: implements a performance counter monitoring utility for memory controller channels.

5.8 pcm-numa.cpp File Reference

Example of using CPU counters: implements a performance counter monitoring utility for NUMA (remote and local memory accesses counting). Example for programming offcore response events.

```
#include <iostream>
#include <unistd.h>
#include <signal.h>
#include <math.h>
#include <iomanip>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <assert.h>
#include "cpucounters.h"
#include "utils.h"
#include <vector>
```

Functions

- void **print_usage** (const char ∗progname)
- template<class StateType >
  void **print_stats** (const StateType &BeforeState, const StateType &AfterState, bool csv)
- int **main** (int argc, char ∗argv[ ])

5.8.1 Detailed Description

Example of using CPU counters: implements a performance counter monitoring utility for NUMA (remote and local memory accesses counting). Example for programming offcore response events.

5.9 pcm-pcie.cpp File Reference

Example of using uncore CBo counters: implements a performance counter monitoring utility for monitoring PCIe bandwidth.

```
#include <iostream>
#include <unistd.h>
#include <signal.h>
#include <math.h>
#include <iomanip>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <assert.h>
#include "cpucounters.h"
#include "utils.h"
```

Classes

- struct **PCIeEvents_t**

Macros

- #define **HACK_TO_REMOVE_DUPLICATE_ERROR**
- #define **NUM_SAMPLES** (1)

Functions

- void **getPCIeEvents** (**PCM** ∗m, PCM::PCIeEventCode opcode, uint32 delay_ms, **PCIeEvents_t** ∗sample)
- void **print_events** ()
- void **print_usage** (const char ∗progname)
- int **main** (int argc, char ∗argv[ ])

Variables

- **PCIeEvents_t aggregate_sample**
- uint32 **num_events** = (sizeof(**PCIeEvents_t**)/sizeof(uint64))
- const uint32 **max_sockets** = 4

5.9.1 Detailed Description

Example of using uncore CBo counters: implements a performance counter monitoring utility for monitoring PCIe bandwidth.

5.10 pcm-sensor.cpp File Reference

Example of using CPU counters: implements a graphical plugin for KDE ksysguard.

```
#include <iostream>
#include <string>
#include <sstream>
#include "cpuasynchcounter.h"
```

Macros

- #define **HACK_TO_REMOVE_DUPLICATE_ERROR**
- #define **OUTPUT_CORE_METRIC**(name, function)
- #define **OUTPUT_SOCKET_METRIC**(name, function)
- #define **OUTPUT_SYSTEM_METRIC**(name, function)

Functions

- int **main** ()

5.10.1 Detailed Description

Example of using CPU counters: implements a graphical plugin for KDE ksysguard.

5.10.2 Macro Definition Documentation

5.10.2.1 #define OUTPUT_CORE_METRIC(*name, function*)

Value:

```
for (uint32 i = 0; i < counters.getNumCores(); ++i) { \
            for (uint32 a = 0; a < counters.getNumSockets(); ++a) \
                if (a == counters.getSocketId(i)) { \
                    stringstream c; \
                    c << "Socket" << a << "/CPU" << i << name; \
                    if (s == c.str()) { \
                        cout << function << endl; \
                    } \
                } \
        }
```

5.10.2.2 #define OUTPUT_SOCKET_METRIC(*name, function*)

Value:

```
for (uint32 i = 0; i < counters.getNumSockets(); ++i) { \
            stringstream c; \
            c << "Socket" << i << name; \
            if (s == c.str()) { \
                cout << function << endl; \
            } \
        }
```

5.10.2.3 #define OUTPUT_SYSTEM_METRIC(*name, function*)

Value:

```
{ \
            stringstream c; \
            c << name; \
            if (s == c.str()) { \
                cout << function << endl; \
            } \
        }
```

5.11 pcm-tsx.cpp File Reference

Example of using CPU counters: implements a performance counter monitoring utility for Intel Transactional Synchronization Extensions.

```
#include <iostream>
#include <unistd.h>
#include <signal.h>
#include <math.h>
#include <iomanip>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <assert.h>
#include "cpucounters.h"
#include "utils.h"
#include <vector>
```

Classes

- struct **TSXEvent**

Functions

- void **print_usage** (const char ∗progname)
- template<class StateType >
  void **print_basic_stats** (const StateType &BeforeState, const StateType &AfterState, bool csv)
- template<class StateType >
  void **print_custom_stats** (const StateType &BeforeState, const StateType &AfterState, bool csv)
- int **findEvent** (const char ∗name)
- int **main** (int argc, char ∗argv[ ])

Variables

- **TSXEvent eventDefinition** [ ]

5.11.1 Detailed Description

Example of using CPU counters: implements a performance counter monitoring utility for Intel Transactional Synchronization Extensions.

5.12 realtime.cpp File Reference

Two use-cases: realtime data structure performance analysis and memory-bandwidth aware scheduling.

```
#include "cpucounters.h"
#include "cpuasynchcounter.h"
#include <iostream>
#include <list>
#include <vector>
#include <algorithm>
#include <sys/time.h>
```

Classes

- struct **T**

Functions

- double **my_timestamp** ()
- long long int **fib** (long long int num)
- void **CPU_intensive_task** ()
- template<class DS >
  void **Memory_intensive_task** (DS &ds)
- double **currentMemoryBandwidth** ()
- template<class DS >
  void **measure** (DS &ds, size_t repeat, size_t nelements)
- int **main** (int argc, char ∗argv[ ])

Variables

- **SystemCounterState before_sstate**
- **SystemCounterState after_sstate**
- double **before_time**
- double **after_time**
- **AsynchronCounterState counters**
- long long int **all_fib** = 0

5.12.1 Detailed Description

Two use-cases: realtime data structure performance analysis and memory-bandwidth aware scheduling.

5.13 types.h File Reference

Internal type and constant definitions.

```
#include <iostream>
#include <istream>
#include <sstream>
#include <iomanip>
```

Classes

- struct **EventSelectRegister**
- struct **FixedEventControlRegister**
- struct **UncoreEventSelectRegister**
- struct **BecktonUncorePMUZDPCTLFVCRegister**
- struct **BecktonUncorePMUCNTCTLRegister**
- struct **MCFGRecord**
- struct **MCFGHeader**

Macros

- #define **COMPILE_FOR_WINDOWS_7**
- #define **INST_RETIRED_ANY_ADDR** (0x309)
- #define **CPU_CLK_UNHALTED_THREAD_ADDR** (0x30A)
- #define **CPU_CLK_UNHALTED_REF_ADDR** (0x30B)
- #define **IA32_CR_PERF_GLOBAL_CTRL** (0x38F)
- #define **IA32_CR_FIXED_CTR_CTRL** (0x38D)
- #define **IA32_PERFEVTSEL0_ADDR** (0x186)
- #define **IA32_PERFEVTSEL1_ADDR** (IA32_PERFEVTSEL0_ADDR + 1)
- #define **IA32_PERFEVTSEL2_ADDR** (IA32_PERFEVTSEL0_ADDR + 2)
- #define **IA32_PERFEVTSEL3_ADDR** (IA32_PERFEVTSEL0_ADDR + 3)
- #define **PERF_MAX_COUNTERS** (7)
- #define **IA32_DEBUGCTL** (0x1D9)
- #define **IA32_PMC0** (0xC1)
- #define **IA32_PMC1** (0xC1 + 1)
- #define **IA32_PMC2** (0xC1 + 2)
- #define **IA32_PMC3** (0xC1 + 3)
- #define **MSR_OFFCORE_RSP0** (0x1A6)
- #define **MSR_OFFCORE_RSP1** (0x1A7)
- #define **PLATFORM_INFO_ADDR** (0xCE)
- #define **IA32_TIME_STAMP_COUNTER** (0x10)
- #define **MEM_LOAD_RETIRED_L3_MISS_EVTNR** (0xCB)
- #define **MEM_LOAD_RETIRED_L3_MISS_UMASK** (0x10)
- #define **MEM_LOAD_RETIRED_L3_UNSHAREDHIT_EVTNR** (0xCB)
- #define **MEM_LOAD_RETIRED_L3_UNSHAREDHIT_UMASK** (0x04)
- #define **MEM_LOAD_RETIRED_L2_HITM_EVTNR** (0xCB)
- #define **MEM_LOAD_RETIRED_L2_HITM_UMASK** (0x08)
- #define **MEM_LOAD_RETIRED_L2_HIT_EVTNR** (0xCB)
- #define **MEM_LOAD_RETIRED_L2_HIT_UMASK** (0x02)
- #define **MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS_EVTNR** (0xD4)
- #define **MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS_UMASK** (0x02)
- #define **MEM_LOAD_UOPS_LLC_HIT_RETIRED_XSNP_NONE_EVTNR** (0xD2)
- #define **MEM_LOAD_UOPS_LLC_HIT_RETIRED_XSNP_NONE_UMASK** (0x08)
- #define **MEM_LOAD_UOPS_LLC_HIT_RETIRED_XSNP_HITM_EVTNR** (0xD2)
- #define **MEM_LOAD_UOPS_LLC_HIT_RETIRED_XSNP_HITM_UMASK** (0x04)
- #define **MEM_LOAD_UOPS_LLC_HIT_RETIRED_XSNP_EVTNR** (0xD2)
- #define **MEM_LOAD_UOPS_LLC_HIT_RETIRED_XSNP_UMASK** (0x07)
- #define **MEM_LOAD_UOPS_RETIRED_L2_HIT_EVTNR** (0xD1)
- #define **MEM_LOAD_UOPS_RETIRED_L2_HIT_UMASK** (0x02)
- #define **ARCH_LLC_REFERENCE_EVTNR** (0x2E)
- #define **ARCH_LLC_REFERENCE_UMASK** (0x4F)
- #define **ARCH_LLC_MISS_EVTNR** (0x2E)
- #define **ARCH_LLC_MISS_UMASK** (0x41)
- #define **ATOM_MEM_LOAD_RETIRED_L2_HIT_EVTNR** (0xCB)
- #define **ATOM_MEM_LOAD_RETIRED_L2_HIT_UMASK** (0x01)
- #define **ATOM_MEM_LOAD_RETIRED_L2_MISS_EVTNR** (0xCB)
- #define **ATOM_MEM_LOAD_RETIRED_L2_MISS_UMASK** (0x02)
- #define **ATOM_MEM_LOAD_RETIRED_L2_HIT_EVTNR** (0xCB)
- #define **ATOM_MEM_LOAD_RETIRED_L2_HIT_UMASK** (0x01)
- #define **ATOM_MEM_LOAD_RETIRED_L2_MISS_EVTNR** (0xCB)
- #define **ATOM_MEM_LOAD_RETIRED_L2_MISS_UMASK** (0x02)
- #define **ATOM_MEM_LOAD_RETIRED_L2_HIT_EVTNR** (0xCB)
- #define **ATOM_MEM_LOAD_RETIRED_L2_HIT_UMASK** (0x01)
- #define **ATOM_MEM_LOAD_RETIRED_L2_MISS_EVTNR** (0xCB)

- #define **ATOM_MEM_LOAD_RETIRED_L2_MISS_UMASK** (0x02)
- #define **MSR_UNCORE_PERF_GLOBAL_CTRL_ADDR** (0x391)
- #define **MSR_UNCORE_PERFEVTSEL0_ADDR** (0x3C0)
- #define **MSR_UNCORE_PERFEVTSEL1_ADDR** (MSR_UNCORE_PERFEVTSEL0_ADDR + 1)
- #define **MSR_UNCORE_PERFEVTSEL2_ADDR** (MSR_UNCORE_PERFEVTSEL0_ADDR + 2)
- #define **MSR_UNCORE_PERFEVTSEL3_ADDR** (MSR_UNCORE_PERFEVTSEL0_ADDR + 3)
- #define **MSR_UNCORE_PERFEVTSEL4_ADDR** (MSR_UNCORE_PERFEVTSEL0_ADDR + 4)
- #define **MSR_UNCORE_PERFEVTSEL5_ADDR** (MSR_UNCORE_PERFEVTSEL0_ADDR + 5)
- #define **MSR_UNCORE_PERFEVTSEL6_ADDR** (MSR_UNCORE_PERFEVTSEL0_ADDR + 6)
- #define **MSR_UNCORE_PERFEVTSEL7_ADDR** (MSR_UNCORE_PERFEVTSEL0_ADDR + 7)
- #define **MSR_UNCORE_PMC0** (0x3B0)
- #define **MSR_UNCORE_PMC1** (MSR_UNCORE_PMC0 + 1)
- #define **MSR_UNCORE_PMC2** (MSR_UNCORE_PMC0 + 2)
- #define **MSR_UNCORE_PMC3** (MSR_UNCORE_PMC0 + 3)
- #define **MSR_UNCORE_PMC4** (MSR_UNCORE_PMC0 + 4)
- #define **MSR_UNCORE_PMC5** (MSR_UNCORE_PMC0 + 5)
- #define **MSR_UNCORE_PMC6** (MSR_UNCORE_PMC0 + 6)
- #define **MSR_UNCORE_PMC7** (MSR_UNCORE_PMC0 + 7)
- #define **UNC_QMC_WRITES_FULL_ANY_EVTNR** (0x2F)
- #define **UNC_QMC_WRITES_FULL_ANY_UMASK** (0x07)
- #define **UNC_QMC_NORMAL_READS_ANY_EVTNR** (0x2C)
- #define **UNC_QMC_NORMAL_READS_ANY_UMASK** (0x07)
- #define **UNC_QHL_REQUESTS_EVTNR** (0x20)
- #define **UNC_QHL_REQUESTS_IOH_READS_UMASK** (0x01)
- #define **UNC_QHL_REQUESTS_IOH_WRITES_UMASK** (0x02)
- #define **UNC_QHL_REQUESTS_REMOTE_READS_UMASK** (0x04)
- #define **UNC_QHL_REQUESTS_REMOTE_WRITES_UMASK** (0x08)
- #define **UNC_QHL_REQUESTS_LOCAL_READS_UMASK** (0x10)
- #define **UNC_QHL_REQUESTS_LOCAL_WRITES_UMASK** (0x20)
- #define **U_MSR_PMON_GLOBAL_CTL** (0x0C00)
- #define **MB0_MSR_PERF_GLOBAL_CTL** (0x0CA0)
- #define **MB0_MSR_PMU_CNT_0** (0x0CB1)
- #define **MB0_MSR_PMU_CNT_CTL_0** (0x0CB0)
- #define **MB0_MSR_PMU_CNT_1** (0x0CB3)
- #define **MB0_MSR_PMU_CNT_CTL_1** (0x0CB2)
- #define **MB0_MSR_PMU_ZDP_CTL_FVC** (0x0CAB)
- #define **MB1_MSR_PERF_GLOBAL_CTL** (0x0CE0)
- #define **MB1_MSR_PMU_CNT_0** (0x0CF1)
- #define **MB1_MSR_PMU_CNT_CTL_0** (0x0CF0)
- #define **MB1_MSR_PMU_CNT_1** (0x0CF3)
- #define **MB1_MSR_PMU_CNT_CTL_1** (0x0CF2)
- #define **MB1_MSR_PMU_ZDP_CTL_FVC** (0x0CEB)
- #define **BB0_MSR_PERF_GLOBAL_CTL** (0x0C20)
- #define **BB0_MSR_PERF_CNT_1** (0x0C33)
- #define **BB0_MSR_PERF_CNT_CTL_1** (0x0C32)
- #define **BB1_MSR_PERF_GLOBAL_CTL** (0x0C60)
- #define **BB1_MSR_PERF_CNT_1** (0x0C73)
- #define **BB1_MSR_PERF_CNT_CTL_1** (0x0C72)
- #define **R_MSR_PMON_CTL0** (0x0E10)
- #define **R_MSR_PMON_CTR0** (0x0E11)
- #define **R_MSR_PMON_CTL1** (0x0E12)
- #define **R_MSR_PMON_CTR1** (0x0E13)
- #define **R_MSR_PMON_CTL2** (0x0E14)
- #define **R_MSR_PMON_CTR2** (0x0E15)
- #define **R_MSR_PMON_CTL3** (0x0E16)

- #define **R_MSR_PMON_CTR3** (0x0E17)
- #define **R_MSR_PMON_CTL4** (0x0E18)
- #define **R_MSR_PMON_CTR4** (0x0E19)
- #define **R_MSR_PMON_CTL5** (0x0E1A)
- #define **R_MSR_PMON_CTR5** (0x0E1B)
- #define **R_MSR_PMON_CTL6** (0x0E1C)
- #define **R_MSR_PMON_CTR6** (0x0E1D)
- #define **R_MSR_PMON_CTL7** (0x0E1E)
- #define **R_MSR_PMON_CTR7** (0x0E1F)
- #define **R_MSR_PMON_CTL8** (0x0E30)
- #define **R_MSR_PMON_CTR8** (0x0E31)
- #define **R_MSR_PMON_CTL9** (0x0E32)
- #define **R_MSR_PMON_CTR9** (0x0E33)
- #define **R_MSR_PMON_CTL10** (0x0E34)
- #define **R_MSR_PMON_CTR10** (0x0E35)
- #define **R_MSR_PMON_CTL11** (0x0E36)
- #define **R_MSR_PMON_CTR11** (0x0E37)
- #define **R_MSR_PMON_CTL12** (0x0E38)
- #define **R_MSR_PMON_CTR12** (0x0E39)
- #define **R_MSR_PMON_CTL13** (0x0E3A)
- #define **R_MSR_PMON_CTR13** (0x0E3B)
- #define **R_MSR_PMON_CTL14** (0x0E3C)
- #define **R_MSR_PMON_CTR14** (0x0E3D)
- #define **R_MSR_PMON_CTL15** (0x0E3E)
- #define **R_MSR_PMON_CTR15** (0x0E3F)
- #define **R_MSR_PORT0_IPERF_CFG0** (0x0E04)
- #define **R_MSR_PORT1_IPERF_CFG0** (0x0E05)
- #define **R_MSR_PORT2_IPERF_CFG0** (0x0E06)
- #define **R_MSR_PORT3_IPERF_CFG0** (0x0E07)
- #define **R_MSR_PORT4_IPERF_CFG0** (0x0E08)
- #define **R_MSR_PORT5_IPERF_CFG0** (0x0E09)
- #define **R_MSR_PORT6_IPERF_CFG0** (0x0E0A)
- #define **R_MSR_PORT7_IPERF_CFG0** (0x0E0B)
- #define **R_MSR_PORT0_IPERF_CFG1** (0x0E24)
- #define **R_MSR_PORT1_IPERF_CFG1** (0x0E25)
- #define **R_MSR_PORT2_IPERF_CFG1** (0x0E26)
- #define **R_MSR_PORT3_IPERF_CFG1** (0x0E27)
- #define **R_MSR_PORT4_IPERF_CFG1** (0x0E28)
- #define **R_MSR_PORT5_IPERF_CFG1** (0x0E29)
- #define **R_MSR_PORT6_IPERF_CFG1** (0x0E2A)
- #define **R_MSR_PORT7_IPERF_CFG1** (0x0E2B)
- #define **R_MSR_PMON_GLOBAL_CTL_7_0** (0x0E00)
- #define **R_MSR_PMON_GLOBAL_CTL_15_8** (0x0E20)
- #define **W_MSR_PMON_GLOBAL_CTL** (0xC80)
- #define **W_MSR_PMON_FIXED_CTR_CTL** (0x395)
- #define **W_MSR_PMON_FIXED_CTR** (0x394)
- #define **MSR_PKG_ENERGY_STATUS** (0x611)
- #define **MSR_RAPL_POWER_UNIT** (0x606)
- #define **MSR_PKG_POWER_INFO** (0x614)
- #define **PCM_INTEL_PCI_VENDOR_ID** (0x8086)
- #define **PCM_PCI_VENDOR_ID_OFFSET** (0)
- #define **JKTIVT_MC0_CH0_REGISTER_DEV_ADDR** (16)
- #define **JKTIVT_MC0_CH1_REGISTER_DEV_ADDR** (16)
- #define **JKTIVT_MC0_CH2_REGISTER_DEV_ADDR** (16)
- #define **JKTIVT_MC0_CH3_REGISTER_DEV_ADDR** (16)

- #define **JKTIVT_MC0_CH0_REGISTER_FUNC_ADDR** (4)
- #define **JKTIVT_MC0_CH1_REGISTER_FUNC_ADDR** (5)
- #define **JKTIVT_MC0_CH2_REGISTER_FUNC_ADDR** (0)
- #define **JKTIVT_MC0_CH3_REGISTER_FUNC_ADDR** (1)
- #define **JKTIVT_MC1_CH0_REGISTER_DEV_ADDR** (30)
- #define **JKTIVT_MC1_CH1_REGISTER_DEV_ADDR** (30)
- #define **JKTIVT_MC1_CH2_REGISTER_DEV_ADDR** (30)
- #define **JKTIVT_MC1_CH3_REGISTER_DEV_ADDR** (30)
- #define **JKTIVT_MC1_CH0_REGISTER_FUNC_ADDR** (4)
- #define **JKTIVT_MC1_CH1_REGISTER_FUNC_ADDR** (5)
- #define **JKTIVT_MC1_CH2_REGISTER_FUNC_ADDR** (0)
- #define **JKTIVT_MC1_CH3_REGISTER_FUNC_ADDR** (1)
- #define **MC_CH_PCI_PMON_BOX_CTL_ADDR** (0x0F4)
- #define **MC_CH_PCI_PMON_FIXED_CTL_ADDR** (0x0F0)
- #define **MC_CH_PCI_PMON_CTL3_ADDR** (0x0E4)
- #define **MC_CH_PCI_PMON_CTL2_ADDR** (0x0E0)
- #define **MC_CH_PCI_PMON_CTL1_ADDR** (0x0DC)
- #define **MC_CH_PCI_PMON_CTL0_ADDR** (0x0D8)
- #define **MC_CH_PCI_PMON_FIXED_CTR_ADDR** (0x0D0)
- #define **MC_CH_PCI_PMON_CTR3_ADDR** (0x0B8)
- #define **MC_CH_PCI_PMON_CTR2_ADDR** (0x0B0)
- #define **MC_CH_PCI_PMON_CTR1_ADDR** (0x0A8)
- #define **MC_CH_PCI_PMON_CTR0_ADDR** (0x0A0)
- #define **JKTIVT_QPI_PORT0_REGISTER_DEV_ADDR** (8)
- #define **JKTIVT_QPI_PORT0_REGISTER_FUNC_ADDR** (2)
- #define **JKTIVT_QPI_PORT1_REGISTER_DEV_ADDR** (9)
- #define **JKTIVT_QPI_PORT1_REGISTER_FUNC_ADDR** (2)
- #define **JKTIVT_QPI_PORT2_REGISTER_DEV_ADDR** (24)
- #define **JKTIVT_QPI_PORT2_REGISTER_FUNC_ADDR** (2)
- #define **QPI_PORT0_MISC_REGISTER_DEV_ADDR** (8)
- #define **QPI_PORT0_MISC_REGISTER_FUNC_ADDR** (0)
- #define **Q_P_PCI_PMON_BOX_CTL_ADDR** (0x0F4)
- #define **Q_P_PCI_PMON_CTL3_ADDR** (0x0E4)
- #define **Q_P_PCI_PMON_CTL2_ADDR** (0x0E0)
- #define **Q_P_PCI_PMON_CTL1_ADDR** (0x0DC)
- #define **Q_P_PCI_PMON_CTL0_ADDR** (0x0D8)
- #define **Q_P_PCI_PMON_CTR3_ADDR** (0x0B8)
- #define **Q_P_PCI_PMON_CTR2_ADDR** (0x0B0)
- #define **Q_P_PCI_PMON_CTR1_ADDR** (0x0A8)
- #define **Q_P_PCI_PMON_CTR0_ADDR** (0x0A0)
- #define **QPI_RATE_STATUS_ADDR** (0x0D4)
- #define **JKTIVT_PCU_MSR_PMON_CTR3_ADDR** (0x0C39)
- #define **JKTIVT_PCU_MSR_PMON_CTR2_ADDR** (0x0C38)
- #define **JKTIVT_PCU_MSR_PMON_CTR1_ADDR** (0x0C37)
- #define **JKTIVT_PCU_MSR_PMON_CTR0_ADDR** (0x0C36)
- #define **JKTIVT_PCU_MSR_PMON_BOX_FILTER_ADDR** (0x0C34)
- #define **JKTIVT_PCU_MSR_PMON_CTL3_ADDR** (0x0C33)
- #define **JKTIVT_PCU_MSR_PMON_CTL2_ADDR** (0x0C32)
- #define **JKTIVT_PCU_MSR_PMON_CTL1_ADDR** (0x0C31)
- #define **JKTIVT_PCU_MSR_PMON_CTL0_ADDR** (0x0C30)
- #define **JKTIVT_PCU_MSR_PMON_BOX_CTL_ADDR** (0x0C24)
- #define **MC_CH_PCI_PMON_BOX_CTL_RST_CONTROL** (1<<0)
- #define **MC_CH_PCI_PMON_BOX_CTL_RST_COUNTERS** (1<<1)
- #define **MC_CH_PCI_PMON_BOX_CTL_FRZ** (1<<8)
- #define **MC_CH_PCI_PMON_BOX_CTL_FRZ_EN** (1<<16)

- #define **UNCORE_PMON_BOX_CTL_VALID_BITS_MASK** ((1<<17)-1)
- #define **MC_CH_PCI_PMON_FIXED_CTL_RST** (1<<19)
- #define **MC_CH_PCI_PMON_FIXED_CTL_EN** (1<<22)
- #define **MC_CH_PCI_PMON_CTL_EVENT**(x) (x<<0)
- #define **MC_CH_PCI_PMON_CTL_UMASK**(x) (x<<8)
- #define **MC_CH_PCI_PMON_CTL_RST** (1<<17)
- #define **MC_CH_PCI_PMON_CTL_EDGE_DET** (1<<18)
- #define **MC_CH_PCI_PMON_CTL_EN** (1<<22)
- #define **MC_CH_PCI_PMON_CTL_INVERT** (1<<23)
- #define **MC_CH_PCI_PMON_CTL_THRESH**(x) (x<<24UL)
- #define **Q_P_PCI_PMON_BOX_CTL_RST_CONTROL** (1<<0)
- #define **Q_P_PCI_PMON_BOX_CTL_RST_COUNTERS** (1<<1)
- #define **Q_P_PCI_PMON_BOX_CTL_RST_FRZ** (1<<8)
- #define **Q_P_PCI_PMON_BOX_CTL_RST_FRZ_EN** (1<<16)
- #define **Q_P_PCI_PMON_CTL_EVENT**(x) (x<<0)
- #define **Q_P_PCI_PMON_CTL_UMASK**(x) (x<<8)
- #define **Q_P_PCI_PMON_CTL_RST** (1<<17)
- #define **Q_P_PCI_PMON_CTL_EDGE_DET** (1<<18)
- #define **Q_P_PCI_PMON_CTL_EVENT_EXT** (1<<21)
- #define **Q_P_PCI_PMON_CTL_EN** (1<<22)
- #define **Q_P_PCI_PMON_CTL_INVERT** (1<<23)
- #define **Q_P_PCI_PMON_CTL_THRESH**(x) (x<<24UL)
- #define **PCU_MSR_PMON_BOX_FILTER_BAND_0**(x) (x<<0)
- #define **PCU_MSR_PMON_BOX_FILTER_BAND_1**(x) (x<<8)
- #define **PCU_MSR_PMON_BOX_FILTER_BAND_2**(x) (x<<16)
- #define **PCU_MSR_PMON_BOX_FILTER_BAND_3**(x) (x<<24)
- #define **PCU_MSR_PMON_BOX_CTL_RST_CONTROL** (1<<0)
- #define **PCU_MSR_PMON_BOX_CTL_RST_COUNTERS** (1<<1)
- #define **PCU_MSR_PMON_BOX_CTL_FRZ** (1<<8)
- #define **PCU_MSR_PMON_BOX_CTL_FRZ_EN** (1<<16)
- #define **PCU_MSR_PMON_CTL_EVENT**(x) (x<<0)
- #define **PCU_MSR_PMON_CTL_OCC_SEL**(x) (x<<14)
- #define **PCU_MSR_PMON_CTL_RST** (1<<17)
- #define **PCU_MSR_PMON_CTL_EDGE_DET** (1<<18)
- #define **PCU_MSR_PMON_CTL_EXTRA_SEL** (1<<21)
- #define **PCU_MSR_PMON_CTL_EN** (1<<22)
- #define **PCU_MSR_PMON_CTL_INVERT** (1<<23)
- #define **PCU_MSR_PMON_CTL_THRESH**(x) (x<<24UL)
- #define **PCU_MSR_PMON_CTL_OCC_INVERT** (1UL<<30UL)
- #define **PCU_MSR_PMON_CTL_OCC_EDGE_DET** (1UL<<31UL)
- #define **JKT_C0_MSR_PMON_CTR3** 0x0D19
- #define **JKT_C0_MSR_PMON_CTR2** 0x0D18
- #define **JKT_C0_MSR_PMON_CTR1** 0x0D17
- #define **JKT_C0_MSR_PMON_CTR0** 0x0D16
- #define **JKT_C0_MSR_PMON_BOX_FILTER** 0x0D14
- #define **JKT_C0_MSR_PMON_CTL3** 0x0D13
- #define **JKT_C0_MSR_PMON_CTL2** 0x0D12
- #define **JKT_C0_MSR_PMON_CTL1** 0x0D11
- #define **JKT_C0_MSR_PMON_CTL0** 0x0D10
- #define **JKT_C0_MSR_PMON_BOX_CTL** 0x0D04
- #define **JKTIVT_CBO_MSR_STEP** 0x0020
- #define **IVT_C0_MSR_PMON_BOX_FILTER1** 0x0D1A
- #define **CBO_MSR_PMON_BOX_CTL_RST_CONTROL** (1<<0)
- #define **CBO_MSR_PMON_BOX_CTL_RST_COUNTERS** (1<<1)
- #define **CBO_MSR_PMON_BOX_CTL_FRZ** (1<<8)

- #define **CBO_MSR_PMON_BOX_CTL_FRZ_EN** (1<<16)
- #define **CBO_MSR_PMON_CTL_EVENT**(x) (x<<0)
- #define **CBO_MSR_PMON_CTL_UMASK**(x) (x<<8)
- #define **CBO_MSR_PMON_CTL_RST** (1<<17)
- #define **CBO_MSR_PMON_CTL_EDGE_DET** (1<<18)
- #define **CBO_MSR_PMON_CTL_TID_EN** (1<<19)
- #define **CBO_MSR_PMON_CTL_EN** (1<<22)
- #define **CBO_MSR_PMON_CTL_INVERT** (1<<23)
- #define **CBO_MSR_PMON_CTL_THRESH**(x) (x<<24UL)
- #define **JKT_CBO_MSR_PMON_BOX_FILTER_OPC**(x) (x<<23UL)
- #define **IVT_CBO_MSR_PMON_BOX_FILTER1_OPC**(x) (x<<20UL)
- #define **MSR_PACKAGE_THERM_STATUS** (0x01B1)
- #define **MSR_IA32_THERM_STATUS** (0x019C)
- #define **PCM_INVALID_THERMAL_HEADROOM** ((std::numeric_limits<int32>::min)())
- #define **MSR_DRAM_ENERGY_STATUS** (0x0619)
- #define **MSR_PKG_C2_RESIDENCY** (0x60D)
- #define **MSR_PKG_C3_RESIDENCY** (0x3F8)
- #define **MSR_PKG_C6_RESIDENCY** (0x3F9)
- #define **MSR_PKG_C7_RESIDENCY** (0x3FA)
- #define **MSR_CORE_C3_RESIDENCY** (0x3FC)
- #define **MSR_CORE_C6_RESIDENCY** (0x3FD)
- #define **MSR_CORE_C7_RESIDENCY** (0x3FE)

Typedefs

- typedef unsigned long long **uint64**
- typedef signed long long **int64**
- typedef unsigned int **uint32**
- typedef signed int **int32**

Functions

- std::ostream & **operator**<< (std::ostream &o, const **FixedEventControlRegister** &reg)

5.13.1 Detailed Description

Internal type and constant definitions.