CFS Time-Triggered Ethernet Library Software Design Document

Section 5.0

Contents

1 Data Structure Documentation

1.1 CFE_PSP_Exception_ContextDataEntry_t Struct Reference

Exception Context Data Entry.

`#include <cfe_psp_config.h>`

Data Fields

- UINT32 timebase_upper

  Upper 32 bits of timebase as sampled by hook.
- UINT32 timebase_lower

  Lower 32 bits of timebase as sampled by hook.
- int vector

  vector number
- ESFPPC esf

  Exception stack frame.
- UINT64 force64BitAlign

  Force the spe register to 64 bit alignment.
- SPE_CONTEXT fp

  floating point registers

1.1.1 Detailed Description

Exception Context Data Entry.

1.2 CFE_PSP_Exception_LogData Struct Reference

Exception Log Data Struct.

`#include <cfe_psp_exceptionstorage_types.h>`

Data Fields

- uint32 context_id

  a unique ID assigned to this exception entry
- uint32 context_size

  actual size of the "context_info" data
- CFE_PSP_Exception_SysTaskId_t sys_task_id

  the BSP-specific task info (not osal abstracted id)
- CFE_PSP_Exception_ContextDataEntry_t context_info

  Context Info.

1.2.1 Detailed Description

Exception Log Data Struct.

1.3 CFE_PSP_ExceptionStorage Struct Reference

Exception Storage Struct.

`#include <cfe_psp_exceptionstorage_types.h>`

Data Fields

- volatile uint32 NumWritten

  Num Written.
- volatile uint32 NumRead

  Num Read.
- struct CFE_PSP_Exception_LogData Entries [CFE_PSP_MAX_EXCEPTION_ENTRIES]

  Entries.

1.3.1 Detailed Description

Exception Storage Struct.

1.4 CFE_PSP_MemoryBlock_t Struct Reference

Memory Block Type.

`#include <cfe_psp_memory.h>`

Data Fields

- void * BlockPtr

  Block Pointer.
- size_t BlockSize

  Block Size.

1.4.1 Detailed Description

Memory Block Type.

1.5 CFE_PSP_MemTable_t Struct Reference

Memory Table Type.

`#include <cfe_psp_memory.h>`

Data Fields

- uint32 MemoryType

  Memory Type.
- size_t WordSize

  Word Size.
- cpuaddr StartAddr

  Start Address.
- size_t Size

  Size.
- uint32 Attributes

  Attributes.

1.5.1 Detailed Description

Memory Table Type.

1.6 CFE_PSP_ModuleApi_t Struct Reference

Concrete version of the abstract API definition structure.

`#include <cfe_psp_module.h>`

Data Fields

- CFE_PSP_ModuleType_t ModuleType
  Module Type.
- uint32 OperationFlags
  OperationFlags.
- CFE_PSP_ModuleInitFunc_t Init
  Module Initialization Function.

1.6.1 Detailed Description

Concrete version of the abstract API definition structure.

Note:

More API calls may be added for other module types

1.7 CFE_PSP_OS_Task_and_priority_t Struct Reference

Task name and priority of tasks.

`#include <cfe_psp_config.h>`

Data Fields

- const char * VxWorksTaskName
  Pointer to the task name.
- int32 VxWorksTaskPriority
  Task priority from 0 to 255.

1.7.1 Detailed Description

Task name and priority of tasks.

Description:

This structure will be used to build an array of VxWorks tasks. The task priority of each task name in the array will be modified according to the assigned priority.

1.8 CFE_PSP_ReservedMemoryBootRecord_t Struct Reference

Layout of the vxWorks boot record structure.

`#include <cfe_psp_config.h>`

Data Fields

- uint32 bsp_reset_type
  BSP Reset Type.
- uint32 spare1
  Spare 1.
- uint32 spare2
  Spare 2.
- uint32 spare3
  Spare 3.

1.8.1 Detailed Description

Layout of the vxWorks boot record structure.

Description:

This is statically placed at the beginning of system memory (sysMemTop) which should be reserved in the kernel.

1.9 CFE_PSP_ReservedMemoryMap_t Struct Reference

Reserved Memory Map.

`#include <cfe_psp_memory.h>`

Data Fields

- CFE_PSP_ReservedMemoryBootRecord_t * BootPtr
  Pointer to Reserved Memory Boot Record.
- CFE_PSP_ExceptionStorage_t * ExceptionStoragePtr
  Pointer to Exception Storage.
- CFE_PSP_MemoryBlock_t ResetMemory
  Reset Memory.
- CFE_PSP_MemoryBlock_t VolatileDiskMemory
  Voltatile Disk Memory.
- CFE_PSP_MemoryBlock_t CDSMemory
  CDS Memory.
- CFE_PSP_MemoryBlock_t UserReservedMemory
  User Reservded Memory.
- CFE_PSP_MemTable_t SysMemoryTable [CFE_PSP_MEM_TABLE_SIZE]
  The system memory table.

1.9.1 Detailed Description

Reserved Memory Map.

1.9.2 Field Documentation

1.9.2.1 CFE_PSP_MemTable_t CFE_PSP_ReservedMemoryMap_t::SysMemoryTable[CFE_PSP_MEM_TABLE_SIZE]

The system memory table.

Description:

This is the table used for CFE_PSP_MemRangeGet/Set and related ops that allow CFE applications to query the general system memory map.

2 File Documentation

2.1 cfe_psp.h File Reference

Main PSP public API functions.

```
#include "common_types.h"
#include "osapi.h"
```

Macros

- #define CFE_PSP_SOFT_TIMEBASE_NAME "cFS-Master"

  The name of the software/RTOS timebase for general system timers.

Error and return codes

- #define CFE_PSP_SUCCESS (0)

  Success.
- #define CFE_PSP_ERROR (-1)

  Generic Error.
- #define CFE_PSP_INVALID_POINTER (-2)

  Invalid Pointer.
- #define CFE_PSP_ERROR_ADDRESS_MISALIGNED (-3)

  Misaligned Address.
- #define CFE_PSP_ERROR_TIMEOUT (-4)

  Timeout Error.
- #define CFE_PSP_INVALID_INT_NUM (-5)

  Invalid Integer Number.
- #define CFE_PSP_INVALID_MEM_ADDR (-21)

  Invalid Memory Address.
- #define CFE_PSP_INVALID_MEM_TYPE (-22)

  Invalid Memory Type.
- #define CFE_PSP_INVALID_MEM_RANGE (-23)

  Invalid Memory Range.

- #define CFE_PSP_INVALID_MEM_WORDSIZE (-24)
  Invalid Memory Word Size.
- #define CFE_PSP_INVALID_MEM_SIZE (-25)
  Invalid Memory Size.
- #define CFE_PSP_INVALID_MEM_ATTR (-26)
  Invalid Memory Attribute.
- #define CFE_PSP_ERROR_NOT_IMPLEMENTED (-27)
  Not Implemented.
- #define CFE_PSP_INVALID_MODULE_NAME (-28)
  Invalid Module Name.
- #define CFE_PSP_INVALID_MODULE_ID (-29)
  Invalid Module ID.
- #define CFE_PSP_NO_EXCEPTION_DATA (-30)
  No Exception Data.

Definitions for PSP PANIC types

- #define CFE_PSP_PANIC_STARTUP 1
  Startup.
- #define CFE_PSP_PANIC_VOLATILE_DISK 2
  Volatile Disk.
- #define CFE_PSP_PANIC_MEMORY_ALLOC 3
  Memory Allocation.
- #define CFE_PSP_PANIC_NONVOL_DISK 4
  Nonvolatile Disk.
- #define CFE_PSP_PANIC_STARTUP_SEM 5
  Startup Semaphore.
- #define CFE_PSP_PANIC_CORE_APP 6
  Core App.
- #define CFE_PSP_PANIC_GENERAL_FAILURE 7
  Generic Failure.

Macros for the file loader

- #define BUFF_SIZE 256
  Buffer Size.
- #define SIZE_BYTE 1
  Size Byte.
- #define SIZE_HALF 2
  Size Half.
- #define SIZE_WORD 3
  Size Word.

Define Memory Types

- #define CFE_PSP_MEM_RAM 1
  Memory RAM.
- #define CFE_PSP_MEM_EEPROM 2
  Memory EEPROM.
- #define CFE_PSP_MEM_ANY 3
  Memory ANY.
- #define CFE_PSP_MEM_INVALID 4

Memory INVALID.

Define Memory Read/Write Attributes

- #define CFE_PSP_MEM_ATTR_WRITE 0x01
  Memory Attribute Write.
- #define CFE_PSP_MEM_ATTR_READ 0x02
  Memory Attribute Read.
- #define CFE_PSP_MEM_ATTR_READWRITE 0x03
  Memory Attribute ReadWrite.

Define the Memory Word Sizes

- #define CFE_PSP_MEM_SIZE_BYTE 0x01
  Memory Size Byte.
- #define CFE_PSP_MEM_SIZE_WORD 0x02
  Memory Size Word.
- #define CFE_PSP_MEM_SIZE_DWORD 0x04
  Memory Size DoubleWord.

Reset Types

- #define CFE_PSP_RST_TYPE_PROCESSOR 1
- #define CFE_PSP_RST_TYPE_POWERON 2
- #define CFE_PSP_RST_TYPE_MAX 3

Reset Sub-Types

- #define CFE_PSP_RST_SUBTYPE_POWER_CYCLE 1
  Reset caused by power having been removed and restored.
- #define CFE_PSP_RST_SUBTYPE_PUSH_BUTTON 2
  Reset caused by reset button on the board having been pressed.
- #define CFE_PSP_RST_SUBTYPE_HW_SPECIAL_COMMAND 3
  Reset was caused by a reset line having been stimulated by a hardware special command.
- #define CFE_PSP_RST_SUBTYPE_HW_WATCHDOG 4
  Reset was caused by a watchdog timer expiring.
- #define CFE_PSP_RST_SUBTYPE_RESET_COMMAND 5
  Reset was caused by cFE ES processing a Reset Command .
- #define CFE_PSP_RST_SUBTYPE_EXCEPTION 6
  Reset was caused by a Processor Exception.
- #define CFE_PSP_RST_SUBTYPE_UNDEFINED_RESET 7
  Reset was caused in an unknown manner.
- #define CFE_PSP_RST_SUBTYPE_HWDEBUG_RESET 8
  Reset was caused by a JTAG or BDM connection.
- #define CFE_PSP_RST_SUBTYPE_BANKSWITCH_RESET 9
  Reset reverted to a cFE POWERON due to a boot bank switch.
- #define CFE_PSP_RST_SUBTYPE_MAX 10
  Placeholder to indicate 1+ the maximum value that the PSP will ever use.

Functions

- void CFE_PSP_Main (void)

  Main entry-point.
- void CFE_PSP_GetTime (OS_time_t *LocalTime)

  Get time.
- void CFE_PSP_Restart (uint32 resetType)

  Re-start.
- uint32 CFE_PSP_GetRestartType (uint32 *resetSubType)

  Get restart type.
- void CFE_PSP_FlushCaches (uint32 type, void *address, uint32 size)

  Flush memory caches.
- uint32 CFE_PSP_GetProcessorId (void)

  Get the CPU ID.
- uint32 CFE_PSP_GetSpacecraftId (void)

  Get the spacecraft ID.
- const char * CFE_PSP_GetProcessorName (void)

  Get the processor name.
- uint32 CFE_PSP_GetTimerTicksPerSecond (void)

  Get the timer ticks per second.
- uint32 CFE_PSP_GetTimerLow32Rollover (void)

  Get the lower 32-bit roll-over time value.
- void CFE_PSP_Get_Timebase (uint32 *Tbu, uint32 *Tbl)

  Get the timebase values.
- int32 CFE_PSP_GetCDSSize (uint32 *SizeOfCDS)

  Get the size of the Critical Data Store memory area.
- int32 CFE_PSP_WriteToCDS (const void *PtrToDataToWrite, uint32 CDSOffset, uint32 NumBytes)

  Write to the Critical Data Store memory area.
- int32 CFE_PSP_ReadFromCDS (void *PtrToDataToRead, uint32 CDSOffset, uint32 NumBytes)

  Read from the Critical Data Store memory area.
- int32 CFE_PSP_GetResetArea (cpuaddr *PtrToResetArea, uint32 *SizeOfResetArea)

  Get the location and size of the ES Reset memory area.
- int32 CFE_PSP_GetUserReservedArea (cpuaddr *PtrToUserArea, uint32 *SizeOfUserArea)

  Get the location and size of the cFE user-reserved memory area.
- int32 CFE_PSP_GetVolatileDiskMem (cpuaddr *PtrToVolDisk, uint32 *SizeOfVolDisk)

  Get the location and size of the cFE volatile memory area.
- int32 CFE_PSP_GetKernelTextSegmentInfo (cpuaddr *PtrToKernelSegment, uint32 *SizeOfKernelSegment)

  Get the location and size of the kernel text segment.
- int32 CFE_PSP_GetCFETextSegmentInfo (cpuaddr *PtrToCFESegment, uint32 *SizeOfCFESegment)

  Get the location and size of the cFE text segment.
- void CFE_PSP_WatchdogInit (void)

  Initialize the watchdog timer.
- void CFE_PSP_WatchdogEnable (void)

  Enable the watchdog timer.
- void CFE_PSP_WatchdogDisable (void)

  Disable the watchdog timer.
- void CFE_PSP_WatchdogService (void)

Service the watchdog timer.

- uint32 CFE_PSP_WatchdogGet (void)

  Get the watchdog time.
- void CFE_PSP_WatchdogSet (uint32 watchDogValue)

  Set the watchdog time.
- void CFE_PSP_Panic (int32 errorCode)

  Abort cFE startup.
- int32 CFE_PSP_InitSSR (uint32 bus, uint32 device, char *DeviceName)

  Initialize the Solid State Recorder.
- void CFE_PSP_AttachExceptions (void)

  Initialize exception handling.
- void CFE_PSP_SetDefaultExceptionEnvironment (void)

  Initialize default exception handling.
- uint32 CFE_PSP_Exception_GetCount (void)

  Get the exception count.
- int32 CFE_PSP_Exception_GetSummary (uint32 *ContextLogId, osal_id_t *TaskId, char *ReasonBuf, uint32 ReasonSize)

  Translate a stored exception log entry into a summary string.
- int32 CFE_PSP_Exception_CopyContext (uint32 ContextLogId, void *ContextBuf, uint32 ContextSize)

  Translate a stored exception log entry into a summary string.
- int32 CFE_PSP_PortRead8 (cpuaddr PortAddress, uint8 *ByteValue)

  Read one byte from memory.
- int32 CFE_PSP_PortWrite8 (cpuaddr PortAddress, uint8 ByteValue)

  Write one byte to memory.
- int32 CFE_PSP_PortRead16 (cpuaddr PortAddress, uint16 *uint16Value)

  Read two bytes from memory.
- int32 CFE_PSP_PortWrite16 (cpuaddr PortAddress, uint16 uint16Value)

  Write two bytes to memory.
- int32 CFE_PSP_PortRead32 (cpuaddr PortAddress, uint32 *uint32Value)

  Read four bytes from memory.
- int32 CFE_PSP_PortWrite32 (cpuaddr PortAddress, uint32 uint32Value)

  Write four bytes to memory.
- int32 CFE_PSP_MemRead8 (cpuaddr MemoryAddress, uint8 *ByteValue)

  Read an 8-bit value from memory.
- int32 CFE_PSP_MemWrite8 (cpuaddr MemoryAddress, uint8 ByteValue)

  Write an 8-bit value to memory.
- int32 CFE_PSP_MemRead16 (cpuaddr MemoryAddress, uint16 *uint16Value)

  Read an 16-bit value from memory.
- int32 CFE_PSP_MemWrite16 (cpuaddr MemoryAddress, uint16 uint16Value)

  Write 16-bit value to memory.
- int32 CFE_PSP_MemRead32 (cpuaddr MemoryAddress, uint32 *uint32Value)

  Read a 32-bit value from memory.
- int32 CFE_PSP_MemWrite32 (cpuaddr MemoryAddress, uint32 uint32Value)

  Write a 32-bit value to memory.
- int32 CFE_PSP_MemCpy (void *dest, const void *src, uint32 size)

  Copy from one memory block to another memory block.
- int32 CFE_PSP_MemSet (void *dest, uint8 value, uint32 size)

Initialize the specified memory block with the specified value.

- int32 CFE_PSP_MemValidateRange (cpuaddr Address, size_t Size, uint32 MemoryType)

  Validate memory range and type.
- uint32 CFE_PSP_MemRanges (void)

  Get the number of memory ranges.
- int32 CFE_PSP_MemRangeSet (uint32 RangeNum, uint32 MemoryType, cpuaddr StartAddr, size_t Size, size_t WordSize, uint32 Attributes)

  Set an entry in the memory range table.
- int32 CFE_PSP_MemRangeGet (uint32 RangeNum, uint32 *MemoryType, cpuaddr *StartAddr, size_t *Size, size_t *WordSize, uint32 *Attributes)

  Get an entry in the memory range table.
- int32 CFE_PSP_EepromWrite8 (cpuaddr MemoryAddress, uint8 ByteValue)

  Write an 8-bit value to memory.
- int32 CFE_PSP_EepromWrite16 (cpuaddr MemoryAddress, uint16 uint16Value)

  Write a 16-bit value to memory.
- int32 CFE_PSP_EepromWrite32 (cpuaddr MemoryAddress, uint32 uint32Value)

  Write a 32-bit value to memory.
- int32 CFE_PSP_EepromWriteEnable (uint32 Bank)

  Enable EEPROM for write operations.
- int32 CFE_PSP_EepromWriteDisable (uint32 Bank)

  Disable EEPROM from write operations.
- int32 CFE_PSP_EepromPowerUp (uint32 Bank)

  Power on the EEPROM.
- int32 CFE_PSP_EepromPowerDown (uint32 Bank)

  Power down the EEPROM.
- const char * CFE_PSP_GetVersionString (void)

  Obtain the PSP version/baseline identifier string.
- const char * CFE_PSP_GetVersionCodeName (void)

  Obtain the version code name.
- void CFE_PSP_GetVersionNumber (uint8 VersionNumbers[4])

  Obtain the PSP numeric version numbers as uint8 values.
- uint32 CFE_PSP_GetBuildNumber (void)

  Obtain the PSP library numeric build number.

2.1.1 Detailed Description

Main PSP public API functions.

Copyright

Description:

This file contains the cFE Platform Support Package(PSP) prototypes. The PSP functions serve as the "glue" between the RTOS and the cFS.
The functions fill gaps that are not really considered part of the OSAL, but are required for the cFE implementation. It is possible that some of these functions could migrate into the OSAL.

Limitations, Assumptions, External Events, and Notes:

None

2.1.2 Macro Definition Documentation

2.1.2.1 #define CFE_PSP_RST_TYPE_MAX 3

Placeholder to indicate 1+ the maximum value that the PSP will ever use.

2.1.2.2 #define CFE_PSP_RST_TYPE_POWERON 2

All memory has been cleared

2.1.2.3 #define CFE_PSP_RST_TYPE_PROCESSOR 1

Volatile disk, Critical Data Store and User Reserved memory could still be valid

2.1.2.4 #define CFE_PSP_SOFT_TIMEBASE_NAME "cFS-Master"

The name of the software/RTOS timebase for general system timers.

This name may be referred to by CFE TIME and/or SCH when setting up its own timers.

2.1.3 Function Documentation

2.1.3.1 void CFE_PSP_AttachExceptions (void)

Initialize exception handling.

Description:

This function sets up the exception environment for a particular platform.

Assumptions, External Events, and Notes:

For VxWorks, this function initializes the EDR policy handling. The handler is called for every exception that other handlers do not handle. Note that the floating point exceptions are handled by the default floating point exception handler, which does a graceful recovery from floating point exceptions in the file speExcLib.c.

Parameters

None	

Returns

None

2.1.3.2 int32 CFE_PSP_EepromPowerDown (uint32 *Bank*)

Power down the EEPROM.

Description:

This function powers down the specified EEPROM bank.

Assumptions, External Events, and Notes:

This function is currently not implemented.

Parameters

in	*Bank*	- The EEPROM bank to power down

Returns

CFE_PSP_SUCCESS

2.1.3.3 int32 CFE_PSP_EepromPowerUp (uint32 *Bank*)

Power on the EEPROM.

Description:

This function powers on the specified EEPROM bank.

Assumptions, External Events, and Notes:

This function is currently not implemented.

Parameters

in	*Bank*	- The EEPROM bank to power on

Returns

CFE_PSP_SUCCESS

2.1.3.4 int32 CFE_PSP_EepromWrite16 (cpuaddr *MemoryAddress,* uint16 *uint16Value*)

Write a 16-bit value to memory.

Description:

This function writes a 16-bit value to the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*MemoryAddress*	- The memory address to write to
in	*uint16Value*	- A 16-bit value to be written

Returns

CFE_PSP_SUCCESS - Data wrote successfully
CFE_PSP_ERROR_ADDRESS_MISALIGNED - The Address is not aligned to 16-bit addressing scheme.

2.1.3.5 int32 CFE_PSP_EepromWrite32 (cpuaddr *MemoryAddress,* uint32 *uint32Value*)

Write a 32-bit value to memory.

Description:

This function writes a 32-bit value to the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*MemoryAddress*	- The memory address to write to
in	*uint32Value*	- A 32-bit value to be written

Returns

CFE_PSP_SUCCESS - Data wrote successfully
CFE_PSP_ERROR_ADDRESS_MISALIGNED - The Address is not aligned to 16-bit addressing scheme.

2.1.3.6 int32 CFE_PSP_EepromWrite8 (cpuaddr *MemoryAddress,* uint8 *ByteValue*)

Write an 8-bit value to memory.

Description:

This function writes an 8-bit value to the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*MemoryAddress*	- The memory address to write to
in	*ByteValue*	- An 8-bit value to be written

Returns

CFE_PSP_SUCCESS - Data wrote successfully
CFE_PSP_ERROR_ADDRESS_MISALIGNED - The Address is not aligned to 16-bit addressing scheme.

2.1.3.7 int32 CFE_PSP_EepromWriteDisable (uint32 *Bank*)

Disable EEPROM from write operations.

Description:

This function disables the specified EEPROM bank from write operations.

Assumptions, External Events, and Notes:

This function is currently not implemented.

Parameters

in	*Bank*	- The EEPROM bank to disable

Returns

CFE_PSP_SUCCESS

2.1.3.8 int32 CFE_PSP_EepromWriteEnable (uint32 *Bank*)

Enable EEPROM for write operations.

Description:

This function enables the specified EEPROM bank for write operations.

Assumptions, External Events, and Notes:

This function is currently not implemented.

Parameters

in	*Bank*	- The EEPROM bank to enable

Returns

CFE_PSP_SUCCESS

2.1.3.9 int32 CFE_PSP_Exception_CopyContext (uint32 *ContextLogId,* void * *ContextBuf,* uint32 *ContextSize*)

Translate a stored exception log entry into a summary string.

Description:

This function takes a stored exception-log entry and converts it into a summary string.

Assumptions, External Events, and Notes:

None

Parameters

in	*ContextLogId*	- The stored exception log ID
out	*ContextBuf*	- Pointer to the variable that stores the copied data
out	*ContextSize*	- The maximum length of the buffer, ContextBuf

Returns

The actual size of the copied data

2.1.3.10 uint32 CFE_PSP_Exception_GetCount (void)

Get the exception count.

Description:

This function fetches the exception count.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

The exception count

2.1.3.11 int32 CFE_PSP_Exception_GetSummary (uint32 * *ContextLogId,* osal_id_t * *TaskId,* char * *ReasonBuf,* uint32 *ReasonSize*)

Translate a stored exception log entry into a summary string.

Description:

This function takes a stored exception-log entry and converts it into a summary string.

Assumptions, External Events, and Notes:

None

Parameters

out	*ContextLogId*	- Pointer to the variable that stores the returned log ID
out	*TaskId*	- Pointer to the variable that stores the returned OSAL task ID
out	*ReasonBuf*	- The buffer that stores the returned string
out	*ReasonSize*	- The maximum length of the buffer, ReasonBuf

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.1.3.12 void CFE_PSP_FlushCaches (uint32 *type,* void * *address,* uint32 *size*)

Flush memory caches.

Description:

This function flushes the processor caches. This function is in the PSP because it is sometimes implemented in hardware and sometimes taken care of by the OS.

Assumptions, External Events, and Notes:

This function is not implemented for the SP0-vxworks6.9 PSP since it is managed by the SP0 BSP/VxWorks OS.

Parameters

in	*type*	- Cache memory type
in	*address*	- Pointer to the cache memory address
in	*size*	- Cache memory size

Returns

None

2.1.3.13 void CFE_PSP_Get_Timebase (uint32 * *Tbu,* uint32 * *Tbl*)

Get the timebase values.

Description:

This function provides the time values of the 32-bit upper and lower registers.

Assumptions, External Events, and Notes:

This function is in the BSP because it is sometimes implemented in hardware and sometimes taken care of by the OS.

Parameters

out	*Tbu*	- Pointer to the returned value of the 32-bit upper register
out	*Tbl*	- Pointer to the returned value of the 32-bit lower register

Returns

None

2.1.3.14 uint32 CFE_PSP_GetBuildNumber (void)

Obtain the PSP library numeric build number.

Description:

The build number is a monotonically increasing number that (coarsely) reflects the number of commits/changes that have been merged since the epoch release. During development cycles this number should increase after each subsequent merge/modification.
Like other version information, this is a fixed number assigned at compile time.

Assumptions, External Events, and Notes:

None

Returns

The OSAL library build number

2.1.3.15 int32 CFE_PSP_GetCDSSize (uint32 * *SizeOfCDS*)

Get the size of the Critical Data Store memory area.

Description:

This function fetches the size of the OS Critical Data Store memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*SizeOfCDS*	- Pointer to the variable that stores the returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.1.3.16 int32 CFE_PSP_GetCFETextSegmentInfo (cpuaddr * *PtrToCFESegment,* uint32 * *SizeOfCFESegment*)

Get the location and size of the cFE text segment.

Description:

This function returns the location and size of the cFE text segment of the memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToCFE-Segment*	- Pointer to the variable that stores the returned memory address
out	*SizeOfCFE-Segment*	- Pointer to the variable that stores returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.1.3.17 int32 CFE_PSP_GetKernelTextSegmentInfo (cpuaddr * *PtrToKernelSegment,* uint32 * *SizeOfKernelSegment*)

Get the location and size of the kernel text segment.

Description:

This function returns the location and size of the kernel text segment of the memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToKernel-Segment*	- Pointer to the variable that stores the returned memory address
out	*SizeOfKernel-Segment*	- Pointer to the variable that stores returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.1.3.18 uint32 CFE_PSP_GetProcessorId (void)

Get the CPU ID.

Description:

This function returns the CPU ID as pre-fdefined by the cFE for specific board and BSP.

Assumptions, External Events, and Notes:

The macro is defined in cfe_platform_cfg.h.

Parameters

None	

Returns

CFE_PSP_CPU_ID

2.1.3.19 const char* CFE_PSP_GetProcessorName (void)

Get the processor name.

Description:

This function returns the CPU name as pre-defined by the cFE.

Assumptions, External Events, and Notes:

The macro is defined in cfe_platform_cfg.h.

Parameters

None	

Returns

CFE_PSP_CPU_NAME

2.1.3.20 int32 CFE_PSP_GetResetArea (cpuaddr * *PtrToResetArea,* uint32 * *SizeOfResetArea*)

Get the location and size of the ES Reset memory area.

Description:

This function returns the location and size of the ES Reset memory area. This area is preserved during a processor reset and is used to store the ER Log, System Log and reset related variables.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToResetArea*	- Pointer to the variable that stores the returned memory address
out	*SizeOfResetArea*	- Pointer to the variable that stores the returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.1.3.21 uint32 CFE_PSP_GetRestartType (uint32 * *resetSubType*)

Get restart type.

Description:

This function returns the last reset type. If a pointer to a valid memory space is passed in, it returns the reset sub-type in that memory. Right now the reset types are application-specific. For the cFE, they are defined in the cfe_es.h file.

Assumptions, External Events, and Notes:

None

Parameters

out	*resetSubType*	- Pointer to the variable that stores the returned reset sub-type

Returns

Last reset type

2.1.3.22 uint32 CFE_PSP_GetSpacecraftId (void)

Get the spacecraft ID.

Description:

This function returns the spacecraft ID as pre-defined by the cFE.

Assumptions, External Events, and Notes:

The macro is defined in cfe_platform_cfg.h.

Parameters

None	

Returns

CFE_PSP_SPACECRAFT_ID

2.1.3.23 void CFE_PSP_GetTime (OS_time_t * *LocalTime*)

Get time.

Description:

This function gets the local time from the hardware on the Vxworks system on the MCP750s. On the other OS/HW setup, it will get time the standard way.

Assumptions, External Events, and Notes:

None

Parameters

out	*LocalTime*	- Pointer to the structure that stores the returned time value

Returns

None

2.1.3.24 uint32 CFE_PSP_GetTimerLow32Rollover (void)

Get the lower 32-bit roll-over time value.

Description:

This function provides the number that the least significant 32-bit of the 64-bit timestamp returned by CFE_PSP_-Get_Timebase() rolls over.

Assumptions, External Events, and Notes:

If the lower 32-bits rolls at 1 second, then the CFE_PSP_TIMER_LOW32_ROLLOVER will be 1000000. If the lower 32-bits rolls at its maximum value (2^32) then CFE_PSP_TIMER_LOW32_ROLLOVER will be 0.

Parameters

None	

Returns

The lower 32-bit value of the roll-over time value

2.1.3.25 uint32 CFE_PSP_GetTimerTicksPerSecond (void)

Get the timer ticks per second.

Description:

This function provides the resolution of the least significant 32-bit of the 64-bit timestamp, returned by CFE_PSP_-Get_Timebase(), in timer ticks per second.

Assumptions, External Events, and Notes:

The timer resolution for accuracy should not be any slower than 1000000 ticks per second, or 1 microsecond per tick.

Parameters

None	

Returns

Number of timer ticks per second

2.1.3.26 int32 CFE_PSP_GetUserReservedArea (cpuaddr * *PtrToUserArea*, uint32 * *SizeOfUserArea*)

Get the location and size of the cFE user-reserved memory area.

Description:

This function returns the location and size of the cFE user-reserved memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToUserArea*	- Pointer to the variable that stores the returned memory address
out	*SizeOfUserArea*	- Pointer to the variable that stores the returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.1.3.27 const char* CFE_PSP_GetVersionCodeName (void)

Obtain the version code name.

Description:

This retrieves the PSP code name.
This is a compatibility indicator for the overall NASA CFS ecosystem.
All modular components which are intended to interoperate should report the same code name.

Assumptions, External Events, and Notes:

None

Returns

Code name. This is a fixed string and cannot be NULL.

2.1.3.28 void CFE_PSP_GetVersionNumber (uint8 *VersionNumbers[4]*)

Obtain the PSP numeric version numbers as uint8 values.

Description:

This retrieves the numeric PSP version identifier as an array of 4 uint8 values.
The array of numeric values is in order of precedence: [0] = Major Number [1] = Minor Number [2] = Revision Number [3] = Mission Revision
The "Mission Revision" (last output) also indicates whether this is an official release, a patched release, or a development version. 0 indicates an official release 1-254 local patch level (reserved for mission use) 255 indicates a development build

Assumptions, External Events, and Notes:

None

Parameters

out	*VersionNumbers*	A fixed-size array to be filled with the version numbers

Returns

None

2.1.3.29 const char* CFE_PSP_GetVersionString (void)

Obtain the PSP version/baseline identifier string.

Description:

This retrieves the PSP version identifier string without extra info.

Assumptions, External Events, and Notes:

None

Returns

Version string. This is a fixed string and cannot be NULL.

2.1.3.30 int32 CFE_PSP_GetVolatileDiskMem (cpuaddr * *PtrToVolDisk,* uint32 * *SizeOfVolDisk*)

Get the location and size of the cFE volatile memory area.

Description:

This function returns the location and size of the cFE volatile memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToVolDisk*	- Pointer to the variable that stores the returned memory address
out	*SizeOfVolDisk*	- Pointer to the variable that stores the returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.1.3.31 int32 CFE_PSP_InitSSR (uint32 *bus*, uint32 *device*, char * *DeviceName*)

Initialize the Solid State Recorder.

Description:

This function configures and initializes the Solid State Recorder for a particular platform.

Assumptions, External Events, and Notes:

This function is not implemented for the SP0-vxworks6.9 PSP since SSR is not used.

Parameters

in	*bus*	- ATA controller number
in	*device*	- ATA drive number
in	*DeviceName*	- Name of the XBD device to create

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.1.3.32 void CFE_PSP_Main (void)

Main entry-point.

Description:

This function is the entry point that the real time OS calls to start cFS. This function will do any BSP/OS-specific setup, then call the entry point of cFS, which is this function.

Assumptions, External Events, and Notes:

cFE should not call this function. See the description.

Parameters

None	

Returns

None

2.1.3.33 int32 CFE_PSP_MemCpy (void * *dest*, const void * *src*, uint32 *size*)

Copy from one memory block to another memory block.

Description:

Copies 'size' byte from memory address pointed by 'src' to memory address pointed by ' dst' For now we are using the standard c library call 'memcpy' but if we find we need to make it more efficient then we'll implement it in assembly.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*dest*	- Pointer to an address to copy to
in,out	*src*	- Pointer address to copy from
in	*size*	- Number of bytes to copy

Returns

CFE_PSP_SUCCESS

2.1.3.34 int32 CFE_PSP_MemRangeGet (uint32 *RangeNum*, uint32 * *MemoryType*, cpuaddr * *StartAddr*, size_t * *Size*, size_t * *WordSize*, uint32 * *Attributes*)

Get an entry in the memory range table.

Description:

This function retrieves an entry in the global CFE_PSP_MemoryTable.

Assumptions, External Events, and Notes:

Because the table is fixed size, the entries are set by using the integer index.

Parameters

in	*RangeNum*	- A 32-bit integer (starting with 0) specifying the MemoryTable entry.
out	*MemoryType*	- A pointer to the 32-bit integer where the Memory Type is stored. Any defined CFE_PSP_MEM_* enumeration can be specified
out	*StartAddr*	- A pointer to the 32-bit integer where the 32-bit starting address of the memory range is stored.
out	*Size*	- A pointer to the 32-bit integer where the 32-bit size of the memory range is stored.
out	*WordSize*	- A pointer to the 32-bit integer where the the minimum addressable size of the range: (CFE_PSP_MEM_SIZE_BYTE, CFE_PSP_MEM_SIZE_WORD, CFE_PSP_MEM_SIZE_DWORD) is stored.

out	*Attributes*	- A pointer to the 32-bit integer where the attributes of the memory range: (CFE_PSP_MEM_ATTR_WRITE, CFE_PSP_MEM_ATTR_READ, CFE_PSP_MEM_ATTR_READWRITE) are stored.

Returns

CFE_PSP_SUCCESS - Memory range returned successfuly
CFE_PSP_INVALID_POINTER - Parameter error
CFE_PSP_INVALID_MEM_RANGE - The index into the table is invalid

2.1.3.35 uint32 CFE_PSP_MemRanges (void)

Get the number of memory ranges.

Description:

This function fetches the number of memory ranges from the global CFE_PSP_MemoryTable.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

The number of entries in the CFE_PSP_MemoryTable

2.1.3.36 int32 CFE_PSP_MemRangeSet (uint32 *RangeNum,* uint32 *MemoryType,* cpuaddr *StartAddr,* size_t *Size,* size_t *WordSize,* uint32 *Attributes*)

Set an entry in the memory range table.

Description:

This function populates an entry in the global CFE_PSP_MemoryTable.

Assumptions, External Events, and Notes:

Because the table is fixed size, the entries are set by using the integer index. No validation is done with the address or size.

Parameters

in	*RangeNum*	- A 32-bit integer (starting with 0) specifying the MemoryTable entry.
in	*MemoryType*	- The memory type to validate, including but not limited to: CFE_PSP_MEM_RAM, CFE_PSP_MEM_EEPROM, or CFE_PSP_MEM_ANY. Any defined CFE_PSP_MEM_∗ enumeration can be specified

in	*StartAddr*	- A 32-bit starting address of the memory range
in	*Size*	- A 32-bit size of the memory range (Address+Size = End Address)
in	*WordSize*	- The minimum addressable size of the range: (CFE_PSP_MEM_SIZE_BYTE, CFE_PSP_MEM_SIZE_WORD, CFE_PSP_MEM_SIZE_DWORD)
in	*Attributes*	- The attributes of the Memory Range: (CFE_PSP_MEM_ATTR_WRITE, CFE_PSP_MEM_ATTR_READ, CFE_PSP_MEM_ATTR_READWRITE)

Returns

CFE_PSP_SUCCESS - Memory range set successfuly
CFE_PSP_INVALID_MEM_RANGE - The index into the table is invalid
CFE_PSP_INVALID_MEM_TYPE - Memory type associated with the range does not match the passed in type.
CFE_PSP_INVALID_MEM_WORDSIZE - The WordSize parameter is not one of the types.
CFE_PSP_INVALID_MEM_ATTR - The Attributes parameter is not one of the predefined types.

2.1.3.37 int32 CFE_PSP_MemRead16 (cpuaddr *MemoryAddress*, uint16 * *uint16Value*)

Read an 16-bit value from memory.

Description:

This function reads a 16-bit value from the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in	*MemoryAddress*	- The memory address to read from
out	*uint16Value*	- Pointer to the variable that stores the 16-bit value read

Returns

CFE_PSP_SUCCESS

2.1.3.38 int32 CFE_PSP_MemRead32 (cpuaddr *MemoryAddress*, uint32 * *uint32Value*)

Read a 32-bit value from memory.

Description:

This function reads a 32-bit value from the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in	*MemoryAddress*	- The memory address to read from
out	*uint32Value*	- Pointer to the variable that stores the 32-bit value read

Returns

CFE_PSP_SUCCESS

2.1.3.39 int32 CFE_PSP_MemRead8 (cpuaddr *MemoryAddress*, uint8 * *ByteValue*)

Read an 8-bit value from memory.

Description:

This function reads an 8-bit value from the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in	*MemoryAddress*	- The memory address to read from
out	*ByteValue*	- Pointer to the variable that stores the 8-bit value read

Returns

CFE_PSP_SUCCESS

2.1.3.40 int32 CFE_PSP_MemSet (void * *dest*, uint8 *value*, uint32 *size*)

Initialize the specified memory block with the specified value.

Description:

Copies 'size' number of byte of value 'value' to memory address pointed by 'dst' .For now we are using the standard c library call 'memset' but if we find we need to make it more efficient then we'll implement it in assembly.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*dest*	- Pointer to destination address
in	*value*	- An 8-bit value to fill in the memory
in	*size*	- The number of values to write

Returns

CFE_PSP_SUCCESS

2.1.3.41 int32 CFE_PSP_MemValidateRange (cpuaddr *Address,* size_t *Size,* uint32 *MemoryType*)

Validate memory range and type.

Description:

This function validates the memory range and type using the global CFE_PSP_MemoryTable.

Assumptions, External Events, and Notes:

None

Parameters

in	*Address*	- A 32-bit starting address of the memory range
in	*Size*	- A 32-bit size of the memory range (Address+Size = End Address)
in	*MemoryType*	- The memory type to validate, including but not limited to: CFE_PSP_MEM_RAM, CFE_PSP_MEM_EEPROM, or CFE_PSP_MEM_ANY. Any defined CFE_PSP_MEM_∗ enumeration can be specified

Returns

CFE_PSP_SUCCESS - Memory range and type information is valid and can be used.
CFE_PSP_INVALID_MEM_ADDR - Starting address is not valid
CFE_PSP_INVALID_MEM_TYPE - Memory type associated with the range does not match the passed in type.
CFE_PSP_INVALID_MEM_RANGE - The Memory range associated with the address is not large enough to contain Address+Size.

2.1.3.42 int32 CFE_PSP_MemWrite16 (cpuaddr *MemoryAddress,* uint16 *uint16Value*)

Write 16-bit value to memory.

Description:

This function writes a 16-bit value to the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*MemoryAddress*	- The memory address to write to
in	*uint16Value*	- A 16-bit value to be written

Returns

CFE_PSP_SUCCESS

2.1.3.43 int32 CFE_PSP_MemWrite32 (cpuaddr *MemoryAddress,* uint32 *uint32Value*)

Write a 32-bit value to memory.

Description:

This function writes a 32-bit value to the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*MemoryAddress*	- The memory address to write to
in	*uint32Value*	- A 32-bit value to be written

Returns

CFE_PSP_SUCCESS

2.1.3.44 int32 CFE_PSP_MemWrite8 (cpuaddr *MemoryAddress*, uint8 *ByteValue*)

Write an 8-bit value to memory.

Description:

This function writes an 8-bit value to the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*MemoryAddress*	- The memory address to write to
in	*ByteValue*	- An 8-bit value to be written

Returns

CFE_PSP_SUCCESS

2.1.3.45 void CFE_PSP_Panic (int32 *errorCode*)

Abort cFE startup.

Description:

This function provides the mechanism to abort the cFE startup process and returns back to the OS.

Assumptions, External Events, and Notes:

This function should not be called by the cFS applications.

Parameters

in	*errorCode*	- Error code that causes the exit

Returns

None

2.1.3.46 int32 CFE_PSP_PortRead16 (cpuaddr *PortAddress*, uint16 * *uint16Value*)

Read two bytes from memory.

Description:

This function reads two bytes from the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in	*PortAddress*	- The port address to read from
out	*uint16Value*	- Pointer to the variable that stores the two-byte value read

Returns

CFE_PSP_SUCCESS

2.1.3.47 int32 CFE_PSP_PortRead32 (cpuaddr *PortAddress*, uint32 * *uint32Value*)

Read four bytes from memory.

Description:

This function reads four bytes from the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in	*PortAddress*	- The port address to read from
out	*uint32Value*	- Pointer to the variable that stores the four-byte value read

Returns

CFE_PSP_SUCCESS

2.1.3.48 int32 CFE_PSP_PortRead8 (cpuaddr *PortAddress*, uint8 * *ByteValue*)

Read one byte from memory.

Description:

This function reads one byte from the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in	*PortAddress*	- The port address to read from
out	*ByteValue*	- Pointer to the variable that stores the one-byte value read

Returns

CFE_PSP_SUCCESS

2.1.3.49 int32 CFE_PSP_PortWrite16 (cpuaddr *PortAddress*, uint16 *uint16Value*)

Write two bytes to memory.

Description:

This function writes two bytes to the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in	*PortAddress*	- The port address to write to
out	*uint16Value*	- Two-byte value to be written

Returns

CFE_PSP_SUCCESS

2.1.3.50 int32 CFE_PSP_PortWrite32 (cpuaddr *PortAddress*, uint32 *uint32Value*)

Write four bytes to memory.

Description:

This function writes four bytes to the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in	*PortAddress*	- The port address to write to
out	*uint32Value*	- Four-byte value to be written

Returns

CFE_PSP_SUCCESS

2.1.3.51 int32 CFE_PSP_PortWrite8 (cpuaddr *PortAddress*, uint8 *ByteValue*)

Write one byte to memory.

Description:

This function writes one byte to the specified memory.

Assumptions, External Events, and Notes:

None

Parameters

in	*PortAddress*	- The port address to write to
out	*ByteValue*	- One-byte value to be written

Returns

CFE_PSP_SUCCESS

2.1.3.52 int32 CFE_PSP_ReadFromCDS (void * *PtrToDataToRead*, uint32 *CDSOffset*, uint32 *NumBytes*)

Read from the Critical Data Store memory area.

Description:

This function reads from the CDS memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToDataToRead*	- Pointer to the data buffer that stores the read data
in	*CDSOffset*	- Memory offset from the beginning of the CDS block
in	*NumBytes*	- Number of bytes to be read

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.1.3.53 void CFE_PSP_Restart (uint32 *resetType*)

Re-start.

Description:

This function is the entry point back to the BSP to restart the processor. cFE calls this function to restart the processor.

Assumptions, External Events, and Notes:

None

Parameters

in	*resetType*	- Type of cFE reset

Returns

None

2.1.3.54 void CFE_PSP_SetDefaultExceptionEnvironment (void)

Initialize default exception handling.

Description:

This function sets up a default exception environment for a particular platform.

Assumptions, External Events, and Notes:

For VxWorks, the exception environment is local to each task. Therefore, this must be called for each task that wants to do floating point and catch exceptions. Currently, this is automatically called from OS_TaskRegister() for every task.

Parameters

None	

Returns

None

2.1.3.55 void CFE_PSP_WatchdogDisable (void)

Disable the watchdog timer.

Description:

This function disables the watchdog timer.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.1.3.56 void CFE_PSP_WatchdogEnable (void)

Enable the watchdog timer.

Description:

This function enables the watchdog timer.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.1.3.57 uint32 CFE_PSP_WatchdogGet (void)

Get the watchdog time.

Description:

This function fetches the watchdog time, in milliseconds.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

The watchdog time in milliseconds

2.1.3.58 void CFE_PSP_WatchdogInit (void)

Initialize the watchdog timer.

Description:

This function configures and intializes the watchdog timer.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.1.3.59 void CFE_PSP_WatchdogService (void)

Service the watchdog timer.

Description:

This function services the watchdog timer according to the value set in CFE_PSP_WatchdogSet().

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.1.3.60 void CFE_PSP_WatchdogSet (uint32 *watchDogValue*)

Set the watchdog time.

Description:

This function sets the current watchdog time, in milliseconds.

Assumptions, External Events, and Notes:

None

Parameters

in	*watchDogValue*	- watchdog time in milliseconds

Returns

None

2.1.3.61 int32 CFE_PSP_WriteToCDS (const void * *PtrToDataToWrite,* uint32 *CDSOffset,* uint32 *NumBytes*)

Write to the Critical Data Store memory area.

Description:

This function write the specified data to the specified memory area of the CDS.

Assumptions, External Events, and Notes:

None

Parameters

in	*PtrToDataToWrite*	- Pointer to the data buffer to be written
in	*CDSOffset*	- Memory offset from the beginning of the CDS block
in	*NumBytes*	- Number of bytes to be written

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.2 cfe_psp_config.h File Reference

Main PSP Configuration File for SP0.

```
#include "common_types.h"
#include <stdio.h>
#include <string.h>
#include <vxWorks.h>
#include <sysLib.h>
#include "speLib.h"
#include "excLib.h"
#include "taskLib.h"
#include "arch/ppc/esfPpc.h"
```

Data Structures

- struct CFE_PSP_ReservedMemoryBootRecord_t

  Layout of the vxWorks boot record structure.
- struct CFE_PSP_Exception_ContextDataEntry_t

  Exception Context Data Entry.
- struct CFE_PSP_OS_Task_and_priority_t

  Task name and priority of tasks.

Macros

- #define CFE_PSP_MEM_TABLE_SIZE 10

  Memory Table Size. This define sets the number of memory ranges that are defined in the memory range defintion table.
- #define CFE_PSP_MAX_EXCEPTION_ENTRIES 4

  Maximum Exception Entries.
- #define CFE_PSP_MEMALIGN_MASK ((cpuaddr)0x1F)

  Memory Alignment Mask.

VxWorks timebase

Description:

The SP0 uses the PowerPC decrementer register. The register is decremented at a speed of:

- *SP0-s DDR2 Configuration: 50 MHz (1/20 = 0.05)*
- *SP0 DDR1 Configuration: 41.666 Mhz (1/24 = 0.041667) For SP0-s the ratio of Denominator/Numerator is 0.05, which is 50 MHz.*

Refer to Aitech 00-0092-01_17_SP0_Programmers_Guide sec. 5.9

Note:

This is expressed as a ratio in case it is not a whole number.

Warning

Numerator needs to be changed to 24 if using with SP0 DDR1.

- #define CFE_PSP_VX_TIMEBASE_PERIOD_NUMERATOR 20

  Numerator.
- #define CFE_PSP_VX_TIMEBASE_PERIOD_DENOMINATOR 1

  Denominator.

Watchdog Settings

- #define CFE_PSP_WATCHDOG_MIN (0)
  Watchdog minimum (in milliseconds)
- #define CFE_PSP_WATCHDOG_MAX (0xFFFFFFFF)
  Watchdog maximum (in milliseconds)

CDS File Location on FLASH

- #define CFE_PSP_CFE_FLASH_FILEPATH "/ffx0/CDS"
  CDS FLASH Memory File Location.

CDS Reading Method Configuration

- #define CFE_PSP_CDS_READ_METHOD_DEFAULT 0
  Default reading method.
- #define CFE_PSP_CDS_READ_METHOD_CRC 1
  CRC reading method.
- #define CFE_PSP_CDS_READ_METHOD_FLASH 2
  Flash reading method.

Memory Scrubbing Configuration

- #define MEMSCRUB_DEFAULT_PRIORITY 250
  Memory Scrub Default Priority.
- #define MEMSCRUB_PRIORITY_UP_RANGE 254
  Memory Scrub Maximum Allowed Priority.
- #define MEMSCRUB_PRIORITY_DOWN_RANGE 120
  Memory Scrub Minimum Allowed Priority.
- #define MEMSCRUB_TASK_NAME "PSPMemScrub"
  Memory Scrub Task Name.

SP0 Info Module

- #define SP0_DATA_DUMP_FILEPATH "/ffx0/PSP_SP0_DUMP"
  SP0 Data Dump Filepath.

Typedefs

- typedef TASK_ID CFE_PSP_Exception_SysTaskId_t
  The data type used by the underlying OS to represent a thread ID.

2.2.1 Detailed Description

Main PSP Configuration File for SP0.

Copyright

Description:

This file includes most of the PSP configuration

Limitations, Assumptions, External Events, and Notes:

None

2.2.2 Macro Definition Documentation

2.2.2.1 #define CFE_PSP_CDS_READ_METHOD_CRC 1

CRC reading method.

Description:

Every CDS writing, a CRC value is saved. This method will verify the CRC everytime reading CDS.

2.2.2.2 #define CFE_PSP_CDS_READ_METHOD_DEFAULT 0

Default reading method.

Description:

Assume the reserved CDS memory is always correct. This will not perform CRC and it will not read from Flash.

Warning

On the SP0, the reserved memory gets erased on reboot.

2.2.2.3 #define CFE_PSP_CDS_READ_METHOD_FLASH 2

Flash reading method.

Description:

This will always read from Flash for every CDS reading.

Warning

Reading from FLASH is considerably slower.

2.2.2.4 #define CFE_PSP_MAX_EXCEPTION_ENTRIES 4

Maximum Exception Entries.

Description:

This define sets the maximum number of exceptions that can be stored. It must always be a power of two.

2.2.2.5 #define CFE_PSP_MEMALIGN_MASK ((cpuaddr)0x1F)

Memory Alignment Mask.

Description:

The alignment to use for each reserved memory block.

This is a mask to be applied to each block base address

Chosen as the cache line size of the SP0 processor (32 bytes) such that the blocks will be cached more efficiently.

2.2.2.6 #define MEMSCRUB_DEFAULT_PRIORITY 250

Memory Scrub Default Priority.

Description:

Set the Active Memory Scrub Task Default Priority

2.2.2.7 #define MEMSCRUB_PRIORITY_DOWN_RANGE 120

Memory Scrub Minimum Allowed Priority.

Description:

Set the Active Memory Scrub Task Down Range Allowable Priority Task Priority can be changed using CFE_PSP_MEM_SCRUB_Set. Down Range priority should not be lower than your apps.

2.2.2.8 #define MEMSCRUB_PRIORITY_UP_RANGE 254

Memory Scrub Maximum Allowed Priority.

Description:

Set the Active Memory Scrub Task Up Range Allowable Priority Task Priority can be changed using CFE_PSP_MEM_SCRUB_Set. Up Range priority is capped by VxWorks OS.

2.2.2.9 #define MEMSCRUB_TASK_NAME "PSPMemScrub"

Memory Scrub Task Name.

Description:

Set the Active Memory Scrub Task Name

2.2.2.10 #define SP0_DATA_DUMP_FILEPATH "/ffx0/PSP_SP0_DUMP"

SP0 Data Dump Filepath.

Description

This file is written only in the case when CFE_PSP_Panic is called.

2.3 cfe_psp_exception.c File Reference

cFE PSP Exception related functions

```
#include <stdio.h>
#include <stddef.h>
#include <string.h>
#include <vxWorks.h>
#include <sysLib.h>
#include "excLib.h"
#include "taskLib.h"
#include "speLib.h"
#include "arch/ppc/vxPpcLib.h"
#include "arch/ppc/esfPpc.h"
#include <private/edrLibP.h>
#include "common_types.h"
#include "osapi.h"
#include "cfe_psp.h"
#include "cfe_psp_config.h"
#include "cfe_psp_exceptionstorage_types.h"
#include "cfe_psp_exceptionstorage_api.h"
#include "cfe_psp_memory.h"
#include <target_config.h>
```

Functions

- STATUS edrErrorPolicyHookRemove (void)

  Declared in Aitech BSP.

Variables

overRideDefaultedrPolicyHandlerHook

- BOOL **overRideDefaultedrPolicyHandlerHook** = FALSE

currentedrPolicyHandlerHook

- LOCAL EDR_POLICY_HANDLER_HOOK **currentedrPolicyHandlerHook** = NULL
- BOOL CFE_PSP_edrPolicyHandlerHook (int type, void ∗pInfo_param, BOOL debug)

  Makes the proper call to CFE_ES_ProcessCoreException.
- void CFE_PSP_AttachExceptions (void)

  Initialize exception handling.
- void CFE_PSP_SetDefaultExceptionEnvironment (void)

  Initialize default exception handling.
- int32 CFE_PSP_ExceptionGetSummary_Impl (const CFE_PSP_Exception_LogData_t ∗Buffer, char ∗ReasonBuf, uint32 ReasonSize)

  Translate the exception context data into a string.

2.3.1 Detailed Description

cFE PSP Exception related functions

Description:

None

Limitations, Assumptions, External Events, and Notes:

The following was found in the `VxWorks 6.9 architecture supplement, pg 179, for PP-C85xx:`
"Do not confuse the hardware floating-point provided by the FPU with that provided by the SPE (see 6.3.10 Signal Processing Engine Support, p.190). If using the e500v2diab or e500v2gnu toolchains, you must use the speSave() speSave() and speRestore() routines to save and restore floating-point context."
The e500 core's SPE is a hardware double precision unit capable of both scalar and vector(SIMD) computation.

2.3.2 Function Documentation

2.3.2.1 void CFE_PSP_AttachExceptions (void)

Initialize exception handling.

Description:

This function sets up the exception environment for a particular platform.

Assumptions, External Events, and Notes:

For VxWorks, this function initializes the EDR policy handling. The handler is called for every exception that other handlers do not handle. Note that the floating point exceptions are handled by the default floating point exception handler, which does a graceful recovery from floating point exceptions in the file speExcLib.c.

Parameters

None	

Returns

None

2.3.2.2 BOOL CFE_PSP_edrPolicyHandlerHook (int *type*, void * *pInfo_param*, BOOL *debug*)

Makes the proper call to CFE_ES_ProcessCoreException.

Description:

Assumptions, External Events, and Notes:

When speSave() is called, it captures the last floating point context, which may not be valid. If a floating point exception occurs you can be almost 100% sure that this will reflect the proper context. But if another type of

exception occurred then this has the possibility of not being valid. Specifically if a task that is not enabled for floating point causes a non-floating point exception, then the meaning of the floating point context will not be valid. If the task is enabled for floating point, then it will be valid.

Parameters

in,out	*type*	- EDR_FACILITY_KERNEL - VxWorks kernel events EDR_FACILITY_INTERRUPT - interrupt handler events EDR_FACILITY_INIT - system startup events EDR_FACILITY_BOOT - system boot events EDR_FACILITY_REBOOT - system restart events EDR_FACILITY_RTP - RTP system events EDR_FACILITY_USER - user generated events
in,out	*pInfo_param*	- A pointer to an architecture-specific EXC_INFO structure, in case of exceptions, with CPU exception information. The exception information is saved by the default VxWorks exception handler. The structure is defined for each architecture in one of these files: target/h/arch/arch/excArchLib.h For example: target/h/arch/ppc/excPpcLib.h
in,out	*debug*	- This flag indicates whether the ED&R system is in debug (also known as lab) mode, or in field (or deployed) mode.

Returns

True - Do not stop offending task
False - Stop offending task

2.3.2.3 int32 CFE_PSP_ExceptionGetSummary_Impl (const CFE_PSP_Exception_LogData_t * *Buffer*, char * *ReasonBuf*, uint32 *ReasonSize*)

Translate the exception context data into a string.

Description:

This function translates the exception context data into a user-friendly "reason" string.

Assumptions, External Events, and Notes:

This is called in an application context to determine the cause of the exception.

Parameters

in	*Buffer*	- Pointer to the Buffer Context data previously stored by ISR/signal handler
out	*ReasonBuf*	- Buffer to store string
in	*ReasonSize*	- Size of string buffer

Returns

CFE_PSP_SUCCESS on success

2.3.2.4 void CFE_PSP_SetDefaultExceptionEnvironment (void)

Initialize default exception handling.

Description:

This function sets up a default exception environment for a particular platform.

Assumptions, External Events, and Notes:

For VxWorks, the exception environment is local to each task. Therefore, this must be called for each task that wants to do floating point and catch exceptions. Currently, this is automatically called from OS_TaskRegister() for every task.

Parameters

None	

Returns

None

2.4 cfe_psp_exceptionstorage.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "common_types.h"
#include "osapi.h"
#include "cfe_psp.h"
#include "cfe_psp_config.h"
#include "cfe_psp_exceptionstorage_types.h"
#include "cfe_psp_exceptionstorage_api.h"
#include "cfe_psp_memory.h"
#include "target_config.h"
```

Macros

CFE_PSP_MAX_EXCEPTION_ENTRY_MASK

- #define **CFE_PSP_MAX_EXCEPTION_ENTRY_MASK** (CFE_PSP_MAX_EXCEPTION_ENTRIES - 1)

CFE_PSP_EXCEPTION_ID_BASE

- #define **CFE_PSP_EXCEPTION_ID_BASE** ((OS_OBJECT_TYPE_USER + 0x101) << OS_OBJECT_TYPE_-SHIFT)
- void CFE_PSP_Exception_Reset (void)

  Reset the exception storage buffer counter.
- CFE_PSP_Exception_LogData_t * CFE_PSP_Exception_GetBuffer (uint32 seq)

  Get the next buffer for exception buffer corresponding to sequence.
- CFE_PSP_Exception_LogData_t * CFE_PSP_Exception_GetNextContextBuffer (void)

  Get the next buffer for exception context storage.
- void CFE_PSP_Exception_WriteComplete (void)

  Wrap up the storage of exception data.
- uint32 CFE_PSP_Exception_GetCount (void)

  Get the exception count.
- int32 CFE_PSP_Exception_GetSummary (uint32 *ContextLogId, osal_id_t *TaskId, char *ReasonBuf, uint32 ReasonSize)

  Translate a stored exception log entry into a summary string.
- int32 CFE_PSP_Exception_CopyContext (uint32 ContextLogId, void *ContextBuf, uint32 ContextSize)

  Translate a stored exception log entry into a summary string.

2.4.1 Detailed Description

```
MCP750 vxWorks 6.2 Version
```

Purpose: cFE PSP Exception related functions.

History: 2007/05/29 A. Cudmore | vxWorks 6.2 MCP750 version 2016/04/07 M.Grubb | Updated for PSP version 1.3

2.4.2 Function Documentation

2.4.2.1 int32 CFE_PSP_Exception_CopyContext (uint32 *ContextLogId,* void ∗ *ContextBuf,* uint32 *ContextSize*)

Translate a stored exception log entry into a summary string.

Description:

This function takes a stored exception-log entry and converts it into a summary string.

Assumptions, External Events, and Notes:

None

Parameters

in	*ContextLogId*	- The stored exception log ID
out	*ContextBuf*	- Pointer to the variable that stores the copied data
out	*ContextSize*	- The maximum length of the buffer, ContextBuf

Returns

The actual size of the copied data

2.4.2.2 CFE_PSP_Exception_LogData_t∗ CFE_PSP_Exception_GetBuffer (uint32 *seq*)

Get the next buffer for exception buffer corresponding to sequence.

Description:

This function obtains a storage buffer corresponding to the given sequence number. The pointer to storage memory is directly returned.

Assumptions, External Events, and Notes:

It is not cleared or modified, and no checks are performed to determine if the sequence number is valid.

Parameters

in	*seq*	- Sequence number

Returns

Pointer to buffer.

2.4.2.3 uint32 CFE_PSP_Exception_GetCount (void)

Get the exception count.

Description:

This function fetches the exception count.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

The exception count

2.4.2.4 CFE_PSP_Exception_LogData_t* CFE_PSP_Exception_GetNextContextBuffer (void)

Get the next buffer for exception context storage.

Description:

This function is invoked by the low level exception handler (typically an ISR/signal) to obtain a buffer for context capture.

Assumptions, External Events, and Notes:

The buffer is cleared (memset zero) before returning to the caller.

Parameters

None	

Returns

Pointer to buffer - If successful
NULL - If storage is full

2.4.2.5 int32 CFE_PSP_Exception_GetSummary (uint32 * *ContextLogId,* osal_id_t * *TaskId,* char * *ReasonBuf,* uint32 *ReasonSize*)

Translate a stored exception log entry into a summary string.

Description:

This function takes a stored exception-log entry and converts it into a summary string.

Assumptions, External Events, and Notes:

None

Parameters

out	*ContextLogId*	- Pointer to the variable that stores the returned log ID
out	*TaskId*	- Pointer to the variable that stores the returned OSAL task ID
out	*ReasonBuf*	- The buffer that stores the returned string
out	*ReasonSize*	- The maximum length of the buffer, ReasonBuf

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.4.2.6 void CFE_PSP_Exception_Reset (void)

Reset the exception storage buffer counter.

Reset the exception storage buffer.

Description:

This function resets the state of exception processing.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.4.2.7 void CFE_PSP_Exception_WriteComplete (void)

Wrap up the storage of exception data.

Description:

This function is invoked by the low level exception handler (typically an ISR/signal) once the exception context capture is complete.

Assumptions, External Events, and Notes:

This should be invoked after a successful call to CFE_PSP_Exception_GetNextContextBuffer() to commit the information to the log.

Parameters

None	

Returns

None

2.5 cfe_psp_exceptionstorage_api.h File Reference

Header file for the PSP exception storage functions.

```
#include "cfe_psp.h"
```

Functions

- struct CFE_PSP_Exception_LogData * CFE_PSP_Exception_GetBuffer (uint32 seq)

  Get the next buffer for exception buffer corresponding to sequence.
- struct CFE_PSP_Exception_LogData * CFE_PSP_Exception_GetNextContextBuffer (void)

  Get the next buffer for exception context storage.
- void CFE_PSP_Exception_WriteComplete (void)

  Wrap up the storage of exception data.
- void CFE_PSP_Exception_Reset (void)

  Reset the exception storage buffer.
- int32 CFE_PSP_ExceptionGetSummary_Impl (const struct CFE_PSP_Exception_LogData *Buffer, char *ReasonBuf, uint32 ReasonSize)

  Translate the exception context data into a string.

2.5.1 Detailed Description

Header file for the PSP exception storage functions.

Copyright

This software was created at NASA's Johnson Space Center. This software is governed by the NASA Open Source Agreement and may be used, distributed and modified only pursuant to the terms of that agreement.

Description:

This file provides a generic storage buffer ring for exceptions and functions to manipulate it.

Limitations, Assumptions, External Events, and Notes:

None

2.5.2 Function Documentation

2.5.2.1 struct CFE_PSP_Exception_LogData* CFE_PSP_Exception_GetBuffer (uint32 *seq*)

Get the next buffer for exception buffer corresponding to sequence.

Description:

This function obtains a storage buffer corresponding to the given sequence number. The pointer to storage memory is directly returned.

Assumptions, External Events, and Notes:

It is not cleared or modified, and no checks are performed to determine if the sequence number is valid.

Parameters

in	*seq*	- Sequence number

Returns

Pointer to buffer.

2.5.2.2 struct CFE_PSP_Exception_LogData* CFE_PSP_Exception_GetNextContextBuffer (void)

Get the next buffer for exception context storage.

Description:

This function is invoked by the low level exception handler (typically an ISR/signal) to obtain a buffer for context capture.

Assumptions, External Events, and Notes:

The buffer is cleared (memset zero) before returning to the caller.

Parameters

None	

Returns

Pointer to buffer - If successful
NULL - If storage is full

2.5.2.3 void CFE_PSP_Exception_Reset (void)

Reset the exception storage buffer.

Description:

This function resets the state of exception processing.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

Reset the exception storage buffer.

Description:

This function resets the state of exception processing.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.5.2.4 void CFE_PSP_Exception_WriteComplete (void)

Wrap up the storage of exception data.

Description:

This function is invoked by the low level exception handler (typically an ISR/signal) once the exception context capture is complete.

Assumptions, External Events, and Notes:

This should be invoked after a successful call to CFE_PSP_Exception_GetNextContextBuffer() to commit the information to the log.

Parameters

None	

Returns

None

2.5.2.5 int32 CFE_PSP_ExceptionGetSummary_Impl (const struct CFE_PSP_Exception_LogData * *Buffer*, char * *ReasonBuf*, uint32 *ReasonSize*)

Translate the exception context data into a string.

Description:

This function translates the exception context data into a user-friendly "reason" string.

Assumptions, External Events, and Notes:

This is called in an application context to determine the cause of the exception.

Parameters

in	*Buffer*	- Pointer to the Buffer Context data previously stored by ISR/signal handler
out	*ReasonBuf*	- Buffer to store string
in	*ReasonSize*	- Size of string buffer

Returns

CFE_PSP_SUCCESS on success

2.6 cfe_psp_exceptionstorage_types.h File Reference

Provides a generic storage buffer ring for exceptions.

```
#include "cfe_psp.h"
#include "cfe_psp_config.h"
```

Data Structures

- struct CFE_PSP_Exception_LogData

  Exception Log Data Struct.
- struct CFE_PSP_ExceptionStorage

  Exception Storage Struct.

Typedefs

- typedef struct CFE_PSP_Exception_LogData CFE_PSP_Exception_LogData_t

  Exception Log Data Type.
- typedef struct CFE_PSP_ExceptionStorage CFE_PSP_ExceptionStorage_t

  Exception Storage Type.

2.6.1 Detailed Description

Provides a generic storage buffer ring for exceptions.

Copyright

Description:

The "MetaData" stores ephemeral exception information which only has meaning within the currently-running process.

This data is important for diagnosing the exception, but it is NOT saved to any persistent log because it will not be relevant once the process ends.

Limitations, Assumptions, External Events, and Notes:

None

2.7 cfe_psp_memory.c File Reference

cFE PSP Memory related functions

```
#include <stdio.h>
#include <string.h>
#include <vxWorks.h>
#include <sysLib.h>
#include <moduleLib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <userReservedMem.h>
#include "common_types.h"
#include "osapi.h"
#include "cfe_psp.h"
#include "cfe_psp_memory.h"
#include <target_config.h>
#include "mem_scrub.h"
#include "psp_mem_scrub.h"
```

Macros

- #define CFE_MODULE_NAME "cfe-core.o"

  Define cFE core loadable module name.

Functions

- unsigned int GetWrsKernelTextStart (void)

  External Kernel Function GetWrsKernelTextStart.
- unsigned int GetWrsKernelTextEnd (void)

  External Kernel Function GetWrsKernelTextEnd.
- int32 CFE_PSP_GetCDSSize (uint32 ∗SizeOfCDS)

  Get the size of the Critical Data Store memory area.
- void CFE_PSP_SetReadCDSMethod (uint8 ucMethod)

  Set the CDS reading method.
- uint8 CFE_PSP_GetReadCDSMethod ()

  Get the CDS reading method.
- void CFE_PSP_SetStaticCRC (uint32 uiNewCRC)

  Change the previous calculated CRC value to new provided value.
- uint32 CFE_PSP_GetStaticCRC (void)

  Get the previous calculated CRC value.
- uint32 CFE_PSP_CalculateCRC (const void ∗DataPtr, uint32 DataLength, uint32 InputCRC)

  Calculate 16 bits CRC from input data.
- int32 CFE_PSP_ReadCDSFromFlash (uint32 ∗puiReadBytes)

  Read the whole CDS data from Flash.
- int32 CFE_PSP_WriteCDSToFlash (uint32 ∗puiWroteBytes)

  Write the whole CDS data on Flash.
- int32 CFE_PSP_WriteToCDS (const void ∗PtrToDataToWrite, uint32 CDSOffset, uint32 NumBytes)

  Write to the Critical Data Store memory area.
- int32 CFE_PSP_ReadFromCDS (void ∗PtrToDataToRead, uint32 CDSOffset, uint32 NumBytes)

Read from the Critical Data Store memory area.

- int32 CFE_PSP_GetResetArea (cpuaddr *PtrToResetArea, uint32 *SizeOfResetArea)

  Get the location and size of the ES Reset memory area.
- int32 CFE_PSP_GetUserReservedArea (cpuaddr *PtrToUserArea, uint32 *SizeOfUserArea)

  Get the location and size of the cFE user-reserved memory area.
- int32 CFE_PSP_GetVolatileDiskMem (cpuaddr *PtrToVolDisk, uint32 *SizeOfVolDisk)

  Get the location and size of the cFE volatile memory area.
- int32 CFE_PSP_InitProcessorReservedMemory (uint32 RestartType)

  Initialize the processor's reserved memory.
- void CFE_PSP_SetupReservedMemoryMap (void)

  Initialize the CFE_PSP_ReservedMemoryMap global object.
- void CFE_PSP_DeleteProcessorReservedMemory (void)

  Delete the processor's reserved memory.
- int32 CFE_PSP_GetKernelTextSegmentInfo (cpuaddr *PtrToKernelSegment, uint32 *SizeOfKernelSegment)

  Get the location and size of the kernel text segment.
- int32 CFE_PSP_GetCFETextSegmentInfo (cpuaddr *PtrToCFESegment, uint32 *SizeOfCFESegment)

  Get the location and size of the cFE text segment.
- void CFE_PSP_MEM_SCRUB_Set (uint32 newStartAddr, uint32 newEndAddr, osal_priority_t task_priority)

  Set the Memory Scrubbing parameters.
- void CFE_PSP_MEM_SCRUB_Delete (void)

  Stop the memory scrubbing task.
- void CFE_PSP_MEM_SCRUB_Status (void)

  Print the Memory Scrubbing statistics.
- void CFE_PSP_MEM_SCRUB_Task (void)

  Main function for the Memory Scrubbing task.
- void CFE_PSP_MEM_SCRUB_Init (void)

  Initialize the Memory Scrubbing task.
- bool CFE_PSP_MEM_SCRUB_isRunning (void)

  Check if the Memory Scrubbing task is running.
- void CFE_PSP_MEM_SCRUB_Enable (void)

  Enable the Memory Scrubbing task.
- void CFE_PSP_MEM_SCRUB_Disable (void)

  Disable the Memory Scrubbing task.

Variables

- char g_cCDSFilename [10] = CFE_PSP_CFE_FLASH_FILEPATH

  CDS File name in File System.
- uint8 g_CDSReadMethod = CFE_PSP_CDS_READ_METHOD_DEFAULT

  CDS Read Method.
- static uint32 sg_uiCDSCrc = 0

  CRC value of CDS content.
- static osal_priority_t sg_uiMemScrubTaskPriority = MEMSCRUB_DEFAULT_PRIORITY

  Task Priority of Memory Scrubbing Task.
- static uint32 sg_uiMemScrubTask_id = 0

  Contains the Active Memory Scrubbing Task ID.
- static uint32 sg_uiMemScrubStartAddr = 0

Contains the Active Memory Scrubbing Start Address.

- static uint32 sg_uiMemScrubEndAddr = 0

  Contains the Active Memory Scrubbing End Address.
- static uint32 sg_uiMemScrubCurrentPage = 0

  Contains the Active Memory Scrubbing Current Page.
- static uint32 sg_uiMemScrubTotalPages = 0

  Contains the Active Memory Scrubbing Total Pages.
- static uint32 sg_endOfRam = 0

  Contains the address of the end of RAM.
- CFE_PSP_ReservedMemoryMap_t CFE_PSP_ReservedMemoryMap

  Pointer to the vxWorks USER_RESERVED_MEMORY area.
- CFE_PSP_MemoryBlock_t PSP_ReservedMemBlock

  Pointer to the reserved memory block.

2.7.1 Detailed Description

cFE PSP Memory related functions

Copyright

Description:

This is the implementation of the cFE memory areas that have to be preserved, and the API that is designed to allow access to them. It also contains memory related routines to return the address of the kernel code used in the cFE checksum.

Limitations, Assumptions, External Events, and Notes:

None

2.7.2 Function Documentation

2.7.2.1 uint32 CFE_PSP_CalculateCRC (const void * *DataPtr,* uint32 *DataLength,* uint32 *InputCRC*)

Calculate 16 bits CRC from input data.

Description:

None

Assumptions, External Events, and Notes:

InputCRC allows the user to calculate the CRC of non-contiguous blocks as a single value. Nominally, the user should set this value to zero.

CFE now includes a function to calculate the CRC. uint32 CFE_ES_CalculateCRC(void *pData, uint32 DataLength, uint32 InputCRC, uint32 TypeCRC); Only CFE_MISSION_ES_CRC_16 is implemented as the TypeCRC

Parameters

in,out	*DataPtr*	- Pointer to the input data buffer
in,out	*DataLength*	- Data buffer length
in,out	*InputCRC*	- A starting value for use in the CRC calculation.

Returns

Calculated CRC value

2.7.2.2 void CFE_PSP_DeleteProcessorReservedMemory (void)

Delete the processor's reserved memory.

Description:

This function unlinks the memory segments within the CFE_PSP_ReservedMemoryMap global object.

Assumptions, External Events, and Notes:

This function is only relevant on systems where the objects are implemented as kernel shared memory segments. The segments will be marked for deletion but the local maps remain usable until the process ends.

Parameters

None	

Returns

None

2.7.2.3 int32 CFE_PSP_GetCDSSize (uint32 * *SizeOfCDS*)

Get the size of the Critical Data Store memory area.

Description:

This function fetches the size of the OS Critical Data Store memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*SizeOfCDS*	- Pointer to the variable that stores the returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.4 int32 CFE_PSP_GetCFETextSegmentInfo (cpuaddr * *PtrToCFESegment*, uint32 * *SizeOfCFESegment*)

Get the location and size of the cFE text segment.

Description:

This function returns the location and size of the cFE text segment of the memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToCFE-Segment*	- Pointer to the variable that stores the returned memory address
out	*SizeOfCFE-Segment*	- Pointer to the variable that stores returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.5 int32 CFE_PSP_GetKernelTextSegmentInfo (cpuaddr * *PtrToKernelSegment*, uint32 * *SizeOfKernelSegment*)

Get the location and size of the kernel text segment.

Description:

This function returns the location and size of the kernel text segment of the memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToKernel-Segment*	- Pointer to the variable that stores the returned memory address
out	*SizeOfKernel-Segment*	- Pointer to the variable that stores returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.6 uint8 CFE_PSP_GetReadCDSMethod ( )

Get the CDS reading method.

Description:

This function get the CDS reading method(use CRC, always read from Flash, or trust the CDS reserved memory in RAM is correct.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

CFE_PSP_CDS_READ_METHOD_DEFAULT - Trust the CDS data on RAM (no CRC or read from Flash)
CFE_PSP_CDS_READ_METHOD_CRC - Check the CRC first then read from Flash if CRC mis-matched
CFE_PSP_CDS_READ_METHOD_FLASH - Always read from Flash

2.7.2.7 int32 CFE_PSP_GetResetArea (cpuaddr * *PtrToResetArea*, uint32 * *SizeOfResetArea*)

Get the location and size of the ES Reset memory area.

Description:

This function returns the location and size of the ES Reset memory area. This area is preserved during a processor reset and is used to store the ER Log, System Log and reset related variables.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToResetArea*	- Pointer to the variable that stores the returned memory address
out	*SizeOfResetArea*	- Pointer to the variable that stores the returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.8 uint32 CFE_PSP_GetStaticCRC (void)

Get the previous calculated CRC value.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

Calculated CRC value

2.7.2.9 int32 CFE_PSP_GetUserReservedArea (cpuaddr * *PtrToUserArea*, uint32 * *SizeOfUserArea*)

Get the location and size of the cFE user-reserved memory area.

Description:

This function returns the location and size of the cFE user-reserved memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToUserArea*	- Pointer to the variable that stores the returned memory address
out	*SizeOfUserArea*	- Pointer to the variable that stores the returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.10 int32 CFE_PSP_GetVolatileDiskMem (cpuaddr * *PtrToVolDisk*, uint32 * *SizeOfVolDisk*)

Get the location and size of the cFE volatile memory area.

Description:

This function returns the location and size of the cFE volatile memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToVolDisk*	- Pointer to the variable that stores the returned memory address
out	*SizeOfVolDisk*	- Pointer to the variable that stores the returned memory size

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.11 int32 CFE_PSP_InitProcessorReservedMemory (uint32 *RestartType*)

Initialize the processor's reserved memory.

Description:

This function initializes all of the memory in the BSP that is preserved on a processor reset. The memory includes the Critical Data Store, the ES Reset Area, the Volatile Disk Memory and the User Reserved Memory.

Assumptions, External Events, and Notes:

This initializes based on the reset type. Typically, the information is preserved on a processor reset, and cleared/reinitialized on a power-on reset.

Parameters

in	*RestartType*	- The reset type

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.12 void CFE_PSP_MEM_SCRUB_Delete (void)

Stop the memory scrubbing task.

Description:

This function deletes the Memory Scrubbing task. The task is deleted and the statistics are reset.

Assumptions, External Events, and Notes:

None

Parameters

-	None

Returns

None

2.7.2.13 void CFE_PSP_MEM_SCRUB_Disable (void)

Disable the Memory Scrubbing task.

Description:

This function disables the Memory Scrubbing task.

Assumptions, External Events, and Notes:

If the task is already running, delete it. If the task is not running, then do nothing.

Parameters

None	

Returns

None

2.7.2.14 void CFE_PSP_MEM_SCRUB_Enable (void)

Enable the Memory Scrubbing task.

Description:

This function enables the Memory Scrubbing task.

Assumptions, External Events, and Notes:

If the task is already running, do nothing. If the task is not running, then start it.

Parameters

None	

Returns

None

2.7.2.15 void CFE_PSP_MEM_SCRUB_Init (void)

Initialize the Memory Scrubbing task.

Description:

This function starts the Memory Scrubbing task as a child thread.

Assumptions, External Events, and Notes:

The scrubMemory function implemented by AiTech may never return an error.

Parameters

None	

Returns

None

2.7.2.16 bool CFE_PSP_MEM_SCRUB_isRunning (void)

Check if the Memory Scrubbing task is running.

Description:

This function provides the status whether the Memory Scrubbing task is running.

Assumptions, External Events, and Notes:

None

Parameters

-	None

Returns

true - If task is running
false - If task is not running

2.7.2.17 void CFE_PSP_MEM_SCRUB_Set (uint32 *newStartAddr,* uint32 *newEndAddr,* osal_priority_t *task_priority*)

Set the Memory Scrubbing parameters.

Description:

This functions set the memory scrubbing parameters.

Assumptions, External Events, and Notes:

After calling this function, the new settings will be applied in the next call to the Activate Memory Scrubbing funtion. If newEndAddr is set to a value larger than the actual physical memory limit, the function will use the phyisical memory limit. Task priority can only be set between MEMSCRUB_PRIORITY_UP_RANGE and MEMSCRUB_PRIORITY_DOWN_RANGE defined in cfe_psp_config.h. Default is set to MEMSCRUB_DEFAULT_PRIORITY.

Parameters

in	*newStartAddr*	- Memory address to start from, usually zero
in	*newEndAddr*	- Memory address to end at, usually end of the physical RAM
in	*task_priority*	- The task priority

Returns

None

2.7.2.18 void CFE_PSP_MEM_SCRUB_Status (void)

Print the Memory Scrubbing statistics.

Description:

This function outputs to the console the following Memory Scrubbing statistics: Start memory address, End memory address, current memory page and total memory pages

Assumptions, External Events, and Notes:

Start memory address is usually 0. End memory address is usually set to the last value of RAM address. Note that a page is 4098 bytes.

Parameters

None	

Returns

None

2.7.2.19 void CFE_PSP_MEM_SCRUB_Task (void)

Main function for the Memory Scrubbing task.

Memory Scrubbing task.

Description:

This is the main function for the Memory Scrubbing task.

Assumptions, External Events, and Notes:

The scrubMemory function implemented by AiTech may never return an error.

Parameters

None	

Returns

None

2.7.2.20 int32 CFE_PSP_ReadCDSFromFlash (uint32 * *puiReadBytes*)

Read the whole CDS data from Flash.

Description:

This function read the whole CDS data on Flash to reserved memory on RAM.

Warning

It took about 117ms to read 131072 bytes (128KB) whole CDS area from Flash.

Assumptions, External Events, and Notes:

None

Parameters

puiReadBytes	- Number of read bytes

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.21 int32 CFE_PSP_ReadFromCDS (void * *PtrToDataToRead,* uint32 *CDSOffset,* uint32 *NumBytes*)

Read from the Critical Data Store memory area.

Description:

This function reads from the CDS memory area.

Assumptions, External Events, and Notes:

None

Parameters

out	*PtrToDataToRead*	- Pointer to the data buffer that stores the read data

in	*CDSOffset*	- Memory offset from the beginning of the CDS block
in	*NumBytes*	- Number of bytes to be read

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.22 void CFE_PSP_SetReadCDSMethod (uint8 *ucMethod*)

Set the CDS reading method.

Description:

This function set the CDS reading method(use CRC, always read from Flash, or trust the CDS reserved memory in RAM is correct.

Assumptions, External Events, and Notes:

None

Parameters

in	*ucMethod*	- Reading method

Returns

None

2.7.2.23 void CFE_PSP_SetStaticCRC (uint32 *uiNewCRC*)

Change the previous calculated CRC value to new provided value.

Description:

This function change the previous calculated CRC value to new provided value. This function is just for testing purpose by forcing the CRC mismatched and read CDS data from Flash.

Assumptions, External Events, and Notes:

None

Parameters

uiNewCRC	- New CRC

Returns

None

2.7.2.24 void CFE_PSP_SetupReservedMemoryMap (void)

Initialize the CFE_PSP_ReservedMemoryMap global object.

Description:

This function initializes the CFE_PSP_ReservedMemoryMap global object.

Assumptions, External Events, and Notes:

This function must be called by the startup code before the map is accessed.

Parameters

None	

Returns

None

2.7.2.25 int32 CFE_PSP_WriteCDSToFlash (uint32 * *puiWroteBytes*)

Write the whole CDS data on Flash.

Description:

This function write the whole CDS data from reserved memory on RAM to Flash.

Assumptions, External Events, and Notes:

It took about 117ms to write 131072 bytes (128KB) whole CDS data to Flash.

Parameters

puiWroteBytes	- Number of written bytes

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.2.26 int32 CFE_PSP_WriteToCDS (const void * *PtrToDataToWrite,* uint32 *CDSOffset,* uint32 *NumBytes*)

Write to the Critical Data Store memory area.

Description:

This function write the specified data to the specified memory area of the CDS.

Assumptions, External Events, and Notes:

None

Parameters

in	*PtrToDataToWrite*	- Pointer to the data buffer to be written
in	*CDSOffset*	- Memory offset from the beginning of the CDS block
in	*NumBytes*	- Number of bytes to be written

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.7.3 Variable Documentation

2.7.3.1 CFE_PSP_ReservedMemoryMap_t CFE_PSP_ReservedMemoryMap

Pointer to the vxWorks USER_RESERVED_MEMORY area.

Map to the reserved memory area(s) Contains a pointer to each of the separate memory blocks.

Description:

The sizes of each memory area is defined in os_processor.h for this architecture.

2.7.3.2 char g_cCDSFilename[10] = CFE_PSP_CFE_FLASH_FILEPATH

CDS File name in File System.

Description:

Fully qualified path of where the CDS file will be stored.

2.7.3.3 uint32 sg_endOfRam = 0 [static]

Contains the address of the end of RAM.

Description:

This variable is filled out once during boot and never changed again. Its value reflects the amount of RAM of the system. When moving cFS from SP0 to SP0-s, the value changes automatically. Value is also used for checking for out of range addresses.

2.7.3.4 uint32 sg_uiMemScrubCurrentPage = 0 [static]

Contains the Active Memory Scrubbing Current Page.

Description:

Current page that the task is working on. This value gets reset whenever task restart.

2.7.3.5 uint32 sg_uiMemScrubEndAddr = 0 [static]

Contains the Active Memory Scrubbing End Address.

Description:

End Address cannot be larger than the maximum RAM

2.7.3.6 uint32 sg_uiMemScrubStartAddr = 0 [static]

Contains the Active Memory Scrubbing Start Address.

Description:

The start address can be anything in the address space.

2.7.3.7 uint32 sg_uiMemScrubTask_id = 0 [static]

Contains the Active Memory Scrubbing Task ID.

Description:

If 0, task is not running

2.7.3.8 uint32 sg_uiMemScrubTotalPages = 0 [static]

Contains the Active Memory Scrubbing Total Pages.

Description:

Total number of pages processed since the start of the task. This value gets reset whenever task restart.

2.8 cfe_psp_memory.h File Reference

Header file for the memory-related supporting functions for the local PSP routines.

```
#include "common_types.h"
#include "cfe_psp_config.h"
#include "cfe_psp_exceptionstorage_types.h"
```

Data Structures

- struct CFE_PSP_MemTable_t
  Memory Table Type.
- struct CFE_PSP_MemoryBlock_t
  Memory Block Type.
- struct CFE_PSP_ReservedMemoryMap_t
  Reserved Memory Map.

Functions

- void CFE_PSP_SetupReservedMemoryMap (void)
  Initialize the CFE_PSP_ReservedMemoryMap global object.
- int32 CFE_PSP_InitProcessorReservedMemory (uint32 RestartType)
  Initialize the processor's reserved memory.
- void CFE_PSP_DeleteProcessorReservedMemory (void)
  Delete the processor's reserved memory.

Variables

- CFE_PSP_ReservedMemoryMap_t CFE_PSP_ReservedMemoryMap
  Map to the reserved memory area(s) Contains a pointer to each of the separate memory blocks.

2.8.1 Detailed Description

Header file for the memory-related supporting functions for the local PSP routines.

Copyright

Description:

None

Limitations, Assumptions, External Events, and Notes:

None

2.8.2 Function Documentation

2.8.2.1 void CFE_PSP_DeleteProcessorReservedMemory (void)

Delete the processor's reserved memory.

Description:

This function unlinks the memory segments within the CFE_PSP_ReservedMemoryMap global object.

Assumptions, External Events, and Notes:

This function is only relevant on systems where the objects are implemented as kernel shared memory segments. The segments will be marked for deletion but the local maps remain usable until the process ends.

Parameters

None	

Returns

None

2.8.2.2 int32 CFE_PSP_InitProcessorReservedMemory (uint32 *RestartType*)

Initialize the processor's reserved memory.

Description:

This function initializes all of the memory in the BSP that is preserved on a processor reset. The memory includes the Critical Data Store, the ES Reset Area, the Volatile Disk Memory and the User Reserved Memory.

Assumptions, External Events, and Notes:

This initializes based on the reset type. Typically, the information is preserved on a processor reset, and cleared/reinitialized on a power-on reset.

Parameters

in	*RestartType*	- The reset type

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.8.2.3 void CFE_PSP_SetupReservedMemoryMap (void)

Initialize the CFE_PSP_ReservedMemoryMap global object.

Description:

This function initializes the CFE_PSP_ReservedMemoryMap global object.

Assumptions, External Events, and Notes:

This function must be called by the startup code before the map is accessed.

Parameters

None	

Returns

None

2.8.3 Variable Documentation

2.8.3.1 CFE_PSP_ReservedMemoryMap_t CFE_PSP_ReservedMemoryMap

Map to the reserved memory area(s) Contains a pointer to each of the separate memory blocks.

Assumptions, External Events, and Notes:

None

Map to the reserved memory area(s) Contains a pointer to each of the separate memory blocks.

Description:

The sizes of each memory area is defined in os_processor.h for this architecture.

2.9 cfe_psp_memrange.c File Reference

```
#include "cfe_psp.h"
#include "cfe_psp_memory.h"
```

Functions

- int32 CFE_PSP_MemValidateRange (cpuaddr Address, size_t Size, uint32 MemoryType)

  Validate memory range and type.
- uint32 CFE_PSP_MemRanges (void)

  Get the number of memory ranges.
- int32 CFE_PSP_MemRangeSet (uint32 RangeNum, uint32 MemoryType, cpuaddr StartAddr, size_t Size, size_t WordSize, uint32 Attributes)

  Set an entry in the memory range table.
- int32 CFE_PSP_MemRangeGet (uint32 RangeNum, uint32 ∗MemoryType, cpuaddr ∗StartAddr, size_t ∗Size, size_t ∗WordSize, uint32 ∗Attributes)

  Get an entry in the memory range table.

2.9.1 Detailed Description

Author : Alan Cudmore

Purpose: This file contains the memory range functions for the cFE Platform Support Package. The memory range is a table of valid memory address ranges maintained by the cFE.

2.9.2 Function Documentation

2.9.2.1 int32 CFE_PSP_MemRangeGet (uint32 *RangeNum*, uint32 ∗ *MemoryType*, cpuaddr ∗ *StartAddr*, size_t ∗ *Size*, size_t ∗ *WordSize*, uint32 ∗ *Attributes*)

Get an entry in the memory range table.

Description:

This function retrieves an entry in the global CFE_PSP_MemoryTable.

Assumptions, External Events, and Notes:

Because the table is fixed size, the entries are set by using the integer index.

Parameters

in	*RangeNum*	- A 32-bit integer (starting with 0) specifying the MemoryTable entry.
out	*MemoryType*	- A pointer to the 32-bit integer where the Memory Type is stored. Any defined CFE_PSP_MEM_∗ enumeration can be specified
out	*StartAddr*	- A pointer to the 32-bit integer where the 32-bit starting address of the memory range is stored.
out	*Size*	- A pointer to the 32-bit integer where the 32-bit size of the memory range is stored.
out	*WordSize*	- A pointer to the 32-bit integer where the the minimum addressable size of the range: (CFE_PSP_MEM_SIZE_BYTE, CFE_PSP_MEM_SIZE_WORD, CFE_PSP_MEM_SIZE_DWORD) is stored.

out	*Attributes*	- A pointer to the 32-bit integer where the attributes of the memory range: (CFE_PSP_MEM_ATTR_WRITE, CFE_PSP_MEM_ATTR_READ, CFE_PSP_MEM_ATTR_READWRITE) are stored.

Returns

CFE_PSP_SUCCESS - Memory range returned successfuly
CFE_PSP_INVALID_POINTER - Parameter error
CFE_PSP_INVALID_MEM_RANGE - The index into the table is invalid

2.9.2.2 uint32 CFE_PSP_MemRanges (void)

Get the number of memory ranges.

Description:

This function fetches the number of memory ranges from the global CFE_PSP_MemoryTable.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

The number of entries in the CFE_PSP_MemoryTable

2.9.2.3 int32 CFE_PSP_MemRangeSet (uint32 *RangeNum,* uint32 *MemoryType,* cpuaddr *StartAddr,* size_t *Size,* size_t *WordSize,* uint32 *Attributes*)

Set an entry in the memory range table.

Description:

This function populates an entry in the global CFE_PSP_MemoryTable.

Assumptions, External Events, and Notes:

Because the table is fixed size, the entries are set by using the integer index. No validation is done with the address or size.

Parameters

in	*RangeNum*	- A 32-bit integer (starting with 0) specifying the MemoryTable entry.
in	*MemoryType*	- The memory type to validate, including but not limited to: CFE_PSP_MEM_RAM, CFE_PSP_MEM_EEPROM, or CFE_PSP_MEM_ANY. Any defined CFE_PSP_MEM_∗ enumeration can be specified

in	*StartAddr*	- A 32-bit starting address of the memory range
in	*Size*	- A 32-bit size of the memory range (Address+Size = End Address)
in	*WordSize*	- The minimum addressable size of the range: (CFE_PSP_MEM_SIZE_BYTE, CFE_PSP_MEM_SIZE_WORD, CFE_PSP_MEM_SIZE_DWORD)
in	*Attributes*	- The attributes of the Memory Range: (CFE_PSP_MEM_ATTR_WRITE, CFE_PSP_MEM_ATTR_READ, CFE_PSP_MEM_ATTR_READWRITE)

Returns

CFE_PSP_SUCCESS - Memory range set successfuly
CFE_PSP_INVALID_MEM_RANGE - The index into the table is invalid
CFE_PSP_INVALID_MEM_TYPE - Memory type associated with the range does not match the passed in type.
CFE_PSP_INVALID_MEM_WORDSIZE - The WordSize parameter is not one of the types.
CFE_PSP_INVALID_MEM_ATTR - The Attributes parameter is not one of the predefined types.

2.9.2.4 int32 CFE_PSP_MemValidateRange (cpuaddr *Address*, size_t *Size*, uint32 *MemoryType*)

Validate memory range and type.

Description:

This function validates the memory range and type using the global CFE_PSP_MemoryTable.

Assumptions, External Events, and Notes:

None

Parameters

in	*Address*	- A 32-bit starting address of the memory range
in	*Size*	- A 32-bit size of the memory range (Address+Size = End Address)
in	*MemoryType*	- The memory type to validate, including but not limited to: CFE_PSP_MEM_RAM, CFE_PSP_MEM_EEPROM, or CFE_PSP_MEM_ANY. Any defined CFE_PSP_MEM_∗ enumeration can be specified

Returns

CFE_PSP_SUCCESS - Memory range and type information is valid and can be used.
CFE_PSP_INVALID_MEM_ADDR - Starting address is not valid
CFE_PSP_INVALID_MEM_TYPE - Memory type associated with the range does not match the passed in type.
CFE_PSP_INVALID_MEM_RANGE - The Memory range associated with the address is not large enough to contain Address+Size.

2.10 cfe_psp_memutils.c File Reference

```
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include "cfe_psp.h"
```

Functions

- int32 CFE_PSP_MemCpy (void ∗dest, const void ∗src, uint32 size)

Copy from one memory block to another memory block.

- int32 CFE_PSP_MemSet (void ∗dest, uint8 value, uint32 size)

  Initialize the specified memory block with the specified value.

2.10.1 Detailed Description

Author : Ezra Yeheskeli

Purpose: This file contains some of the cFE Platform Support Layer. It contains the processor architecture specific calls.

2.10.2 Function Documentation

2.10.2.1 int32 CFE_PSP_MemCpy (void ∗ *dest*, const void ∗ *src*, uint32 *size*)

Copy from one memory block to another memory block.

Description:

Copies 'size' byte from memory address pointed by 'src' to memory address pointed by ' dst' For now we are using the standard c library call 'memcpy' but if we find we need to make it more efficient then we'll implement it in assembly.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*dest*	- Pointer to an address to copy to
in,out	*src*	- Pointer address to copy from
in	*size*	- Number of bytes to copy

Returns

CFE_PSP_SUCCESS

2.10.2.2 int32 CFE_PSP_MemSet (void ∗ *dest*, uint8 *value*, uint32 *size*)

Initialize the specified memory block with the specified value.

Description:

Copies 'size' number of byte of value 'value' to memory address pointed by 'dst' .For now we are using the standard c library call 'memset' but if we find we need to make it more efficient then we'll implement it in assembly.

Assumptions, External Events, and Notes:

None

Parameters

in,out	*dest*	- Pointer to destination address
in	*value*	- An 8-bit value to fill in the memory
in	*size*	- The number of values to write

Returns

CFE_PSP_SUCCESS

2.11 cfe_psp_module.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "osapi.h"
#include "cfe_psp_module.h"
```

CFE PSP Module Base and Index

Description:

When using an OSAL that also supports "opaque object ids", choose values here that will fit in with the OSAL object ID values and not overlap anything.

- #define **CFE_PSP_MODULE_BASE** 0x01100000
- #define **CFE_PSP_MODULE_INDEX_MASK** 0xFFFF
- static uint32 **CFE_PSP_ModuleCount** = 0
- void CFE_PSP_ModuleInitList (CFE_StaticModuleLoadEntry_t ∗ListPtr)

  Initialize a list of Modules.
- void CFE_PSP_ModuleInit (void)

  Initialize a list of Modules.
- int32 CFE_PSP_Module_GetAPIEntry (uint32 PspModuleId, CFE_PSP_ModuleApi_t ∗∗API)

  Get entry point of Module.
- int32 CFE_PSP_Module_FindByName (const char ∗ModuleName, uint32 ∗PspModuleId)

  Find a module by name.

2.11.1 Detailed Description

Created on: Jul 25, 2014 Author: jphickey

2.11.2 Function Documentation

2.11.2.1 int32 CFE_PSP_Module_FindByName (const char ∗ *ModuleName,* uint32 ∗ *PspModuleId*)

Find a module by name.

Obtain the module ID by name.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

in	*ModuleName*	- The name of the Module
in,out	*PspModuleId*	- The Module Id

Returns

CFE_PSP_INVALID_MODULE_NAME
CFE_PSP_SUCCESS

2.11.2.2 int32 CFE_PSP_Module_GetAPIEntry (uint32 *PspModuleId*, CFE_PSP_ModuleApi_t ** *API*)

Get entry point of Module.

Obtain the API for a specific module.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

in	*PspModuleId*	- Module ID
in	*API*	-

Returns

CFE_PSP_INVALID_MODULE_ID
CFE_PSP_SUCCESS

2.11.2.3 void CFE_PSP_ModuleInit (void)

Initialize a list of Modules.

Initialize the included PSP modules.

Description:

Initalize all modules for PSP including user-selected modules

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.11.2.4 void CFE_PSP_ModuleInitList (CFE_StaticModuleLoadEntry_t * *ListPtr*)

Initialize a list of Modules.

Description:

Helper function to initalize a list of modules (not externally called)

Assumptions, External Events, and Notes:

None

Parameters

`in,out`	*ListPtr*	- Pointer to the list of modules

Returns

None

2.12 cfe_psp_module.h File Reference

Header file for the PSP public module data types and functions.

```
#include "cfe_psp.h"
#include "target_config.h"
```

Data Structures

- struct CFE_PSP_ModuleApi_t

  Concrete version of the abstract API definition structure.

Typedefs

- typedef void(* CFE_PSP_ModuleInitFunc_t)(uint32 PspModuleId)

  Protoype for a PSP module initialization function.

Enumerations

- enum CFE_PSP_ModuleType_t {
  CFE_PSP_MODULE_TYPE_INVALID = 0,
  CFE_PSP_MODULE_TYPE_SIMPLE }

  Enum Module Type.

CFE_PSP_MODULE_DECLARE_SIMPLE

Description:

Macro to simplify declaration of the IO Driver API structure according to the required naming convention. The "name" argument should match the name of the module object file

- #define **CFE_PSP_MODULE_DECLARE_SIMPLE**(name)
- CFE_StaticModuleLoadEntry_t CFE_PSP_BASE_MODULE_LIST []

  A list of fixed/base modules associated with the PSP.
- void CFE_PSP_ModuleInit (void)

  Initialize the included PSP modules.
- int32 CFE_PSP_Module_FindByName (const char ∗ModuleName, uint32 ∗PspModuleId)

  Obtain the module ID by name.
- int32 CFE_PSP_Module_GetAPIEntry (uint32 PspModuleId, CFE_PSP_ModuleApi_t ∗∗API)

  Obtain the API for a specific module.

2.12.1 Detailed Description

Header file for the PSP public module data types and functions.

Copyright

This software was created at NASA's Johnson Space Center. This software is governed by the NASA Open Source Agreement and may be used, distributed and modified only pursuant to the terms of that agreement.

Description:

None

Limitations, Assumptions, External Events, and Notes:

None

2.12.2 Macro Definition Documentation

2.12.2.1 #define CFE_PSP_MODULE_DECLARE_SIMPLE(*name*)

Value:

```
static void         name##_Init(uint32 PspModuleId); \
    CFE_PSP_ModuleApi_t CFE_PSP_##name##_API = {         \
        .ModuleType     = CFE_PSP_MODULE_TYPE_SIMPLE,    \
        .OperationFlags = 0,                             \
        .Init           = name##_Init,                   \
    }
```

2.12.3 Enumeration Type Documentation

2.12.3.1 enum CFE_PSP_ModuleType_t

Enum Module Type.

Note:

May be extended in the future

Enumerator

CFE_PSP_MODULE_TYPE_INVALID Type Invalid.

CFE_PSP_MODULE_TYPE_SIMPLE Type Simple.

2.12.4 Function Documentation

2.12.4.1 int32 CFE_PSP_Module_FindByName (const char * *ModuleName*, uint32 * *PspModuleId*)

Obtain the module ID by name.

Description:

This function retrieves the module ID of the given module name.

Assumptions, External Events, and Notes:

Although this is currently prototyped as a function scoped to the PSP, this prototype could be moved to the public area so the cFS could use this.

Parameters

in	*ModuleName*	- Name of the module to look up
out	*PspModuleId*	- Pointer to the variable that stores the returned module ID

Returns

CFE_PSP_SUCCESS
CFE_PSP_INVALID_MODULE_NAME

Obtain the module ID by name.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

in	*ModuleName*	- The name of the Module
in,out	*PspModuleId*	- The Module Id

Returns

CFE_PSP_INVALID_MODULE_NAME
CFE_PSP_SUCCESS

2.12.4.2 int32 CFE_PSP_Module_GetAPIEntry (uint32 *PspModuleId,* CFE_PSP_ModuleApi_t ** *API*)

Obtain the API for a specific module.

Description:

This function retrieves the API structure for a given module ID.

Assumptions, External Events, and Notes:

None

Parameters

in	*PspModuleId*	- The ID of the module (configuration-dependent)
out	*API*	- Pointer to the variable that stores the returned API structure

Returns

CFE_PSP_SUCCESS
CFE_PSP_INVALID_MODULE_ID

Obtain the API for a specific module.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

in	*PspModuleId*	- Module ID
in	*API*	-

Returns

CFE_PSP_INVALID_MODULE_ID
CFE_PSP_SUCCESS

2.12.4.3 void CFE_PSP_ModuleInit (void)

Initialize the included PSP modules.

Description:

This function initializes the include PSP modules.

Assumptions, External Events, and Notes:

This function is an optional part of the PSP and some PSP implementations may not use it.

Note 1: It should only be called during PSP initialization before the system is operational. It is not intended to be called from application code after cFE has started. The function is not necessarily be thread-safe and should be called before any child threads are created.

Note 2: This function does *not* return any status. If a failure occurs during initialization that would make normal operation impossible, then the module itself will call CFE_PSP_Panic() and this will not return. Otherwise, benign/recoverable failures are expected to be just that, and the calling code will not need to take any special action either way. In short, if this function returns, then it means the system is good enough to continue.

Parameters

None	

Returns

None

Initialize the included PSP modules.

Description:

Initalize all modules for PSP including user-selected modules

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.12.5 Variable Documentation

2.12.5.1 CFE_StaticModuleLoadEntry_t CFE_PSP_BASE_MODULE_LIST[ ]

A list of fixed/base modules associated with the PSP.

Description:

This list should be generated by the build system based on the user-selected PSP

2.13 cfe_psp_ntp.c File Reference

API to control NTP Sync.

```
#include "ipcom_err.h"
#include "taskLib.h"
#include <timers.h>
#include "cfe_time_extern_typedefs.h"
#include "cfe_mission_cfg.h"
#include "cfe_psp.h"
#include "cfe_psp_config.h"
#include "cfe_psp_module.h"
#include "psp_time_sync.h"
```

Macros

- #define IP_PORT_VXWORKS 69

VxWorks IPCOM Specific Configuration.

- #define PRE_PRINT_SCOPE "PSP NTP SYNC: "

  Default NTP Sync pre-print string.

NTP Sync Configuration

- #define CFE_MISSION_TIME_SYNC_OS_ENABLE true

  Default NTP Sync Start/Stop on Startup.
- #define CFE_MISSION_TIME_SYNC_OS_SEC 30

  Default Synchronization Frequency.
- #define NTPSYNC_TASK_NAME "PSPNTPSync"

  Default NTP Sync Task Name.
- #define NTPSYNC_DEFAULT_PRIORITY 60

  Default NTP Sync Task Priority.

Functions

- TASK_ID taskNameToId (char *name)

  VxWorks function to get ID of running task.
- IP_PUBLIC Ip_err ipcom_ipd_kill (const char *name)

  VxWorks function to kill a running daemon.
- IP_PUBLIC Ip_err ipcom_ipd_start (const char *name)

  VxWorks function to start a daemon.
- uint32 CFE_TIME_Micro2SubSecs (uint32)

  Convert micro seconds in subseconds.
- void CFE_TIME_SetTime (CFE_TIME_SysTime_t)

  Adjust CFE Time STCF so that local time match the new time.
- CFE_PSP_MODULE_DECLARE_SIMPLE (ntp_clock_vxworks)

  Macro to define this file a PSP Module.
- void ntp_clock_vxworks_Init (uint32 PspModuleId)

  Entry point for the module.
- int32 CFE_PSP_TIME_Init (uint16 timer_frequency_sec)

  Initialize the CFE PSP Time Task synchronizing with the NTP server.
- int32 CFE_PSP_Sync_From_OS_Enable (bool enable)

  Enable/disable time sync.
- bool CFE_PSP_NTP_Daemon_Get_Status (void)

  Get the NTP daemon status.
- int32 net_clock_vxworks_Destroy (void)

  Gracefully shutdown NTP Sync Module.
- int32 CFE_PSP_Sync_From_OS_Freq (uint16 new_frequency_sec)

  Change the sync frequency.
- int32 CFE_PSP_Set_OS_Time (const uint32 ts_sec, const uint32 ts_nsec)

  Set the OS time.
- int32 CFE_PSP_Get_OS_Time (CFE_TIME_SysTime_t *myT)

  Gets the current time from VxWorks OS.
- void CFE_PSP_Update_OS_Time (void)

  Update cFE time.
- int32 CFE_PSP_StartNTPDaemon (void)

Start the NTP client.

- int32 CFE_PSP_StopNTPDaemon (void)

  Stop the NTP client.

- int32 CFE_PSP_NTP_Daemon_Enable (bool enable)

  Enable/disable the NTP client.

Variables

- static uint32 sg_uiPSPNTPTask_id = 0

  Contains the NTP Sync Task ID If 0, task is not running.

- static osal_priority_t sg_uiNTPSyncTaskPriority = NTPSYNC_DEFAULT_PRIORITY

  Current value of NTP Sync priority task.

- static bool sg_bEnableGetTimeFromOS_flag = CFE_MISSION_TIME_SYNC_OS_ENABLE

  Boolean variable to control if to synchronize CFE Time Service with OS local time. True, synch will occur. False, timer will not be disabled, but sync will not execute.

- static uint16 sg_uiOSTimeSync_Sec = CFE_MISSION_TIME_SYNC_OS_SEC

  Change how often to sync CFE Time Service with OS Local Time. OS local time is synchronized to NTP server(s) automatically from within OS if enabled.

2.13.1 Detailed Description

API to control NTP Sync.

Copyright

Purpose:

This file contains the function declaration that synchronize the cFE Time services to the NTP server. Note that the NTP server must be built into the kernel.

Limitations, Assumptions, External Events, and Notes:

The way this module updates the local time is by calling the CFE Time Service function CFE_TIME_SetTime(). The function changes the STCF value.
GSFC developers do not recommend to use this method of updating CFE time, but rather to use the function CFE-_TIME_ExternalTime(). The only way to use this function is by building an app that will periodically (1Hz) get NTP time and publish it via Software Bus.

2.13.2 Macro Definition Documentation

2.13.2.1 #define CFE_MISSION_TIME_SYNC_OS_ENABLE true

Default NTP Sync Start/Stop on Startup.

Description:

Enable or disable the Automatic time sync with the OS

2.13.2.2 #define CFE_MISSION_TIME_SYNC_OS_SEC 30

Default Synchronization Frequency.

Description:

Default number of seconds between time synchronizations. CFE Time Service updates MET and STCF from VxWorks OS. When set to zero, CFE Time will be synchronized only once during start.

Limits

Positive integer up to 255. If this value is too low, it will starve the other processes.

2.13.2.3 #define PRE_PRINT_SCOPE "PSP NTP SYNC: "

Default NTP Sync pre-print string.

Description:

This string is printed before every print related to NTP Sync API.

2.13.3 Function Documentation

2.13.3.1 int32 CFE_PSP_Get_OS_Time (CFE_TIME_SysTime_t * *myT*)

Gets the current time from VxWorks OS.

Description:

This function gets the current VxWorks OS time.

Assumptions, External Events, and Notes:

This function is used by the NTP Sync task to grab the current OS time. It uses CLOCK_REALTIME.

Parameters

in,out	*myT*	- Pointer to the variable that stores the returned time value

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.13.3.2 int32 CFE_PSP_NTP_Daemon_Enable (bool *enable*)

Enable/disable the NTP client.

Description:

This function enables/disables the NTP client task, ipntpd, on VxWorks.

Assumptions, External Events, and Notes:

None

Parameters

in	*enable*	- Boolean flag for enable or disable

Returns

NTP client task ID - If successfully starts the NTP clien task
CFE_PSP_SUCCESS - If successfully stops the NTP client task
CFE_PSP_ERROR

2.13.3.3 bool CFE_PSP_NTP_Daemon_Get_Status (void)

Get the NTP daemon status.

Description:

This function checks if the VxWorks NTP client task is running. It does not check if the task has successfully synchronized with an NTP server.

Assumptions, External Events, and Notes:

The task name for the VxWorks NTP client is the default "ipntpd".

Parameters

None	

Returns

True - If NTP client task is running
False - If NTP client task is not running

2.13.3.4 int32 CFE_PSP_Set_OS_Time (const uint32 *ts_sec*, const uint32 *ts_nsec*)

Set the OS time.

Description:

This function sets the VxWorks OS time.

Assumptions, External Events, and Notes:

The changes do not occur if the NTP client is setup to synchrone with an NTP server. Set the OS CLOCK_REALTIME to a specified timestamp. Parameters are in UNIX time format, since Epoch 1/1/1970.

Parameters

in	*ts_sec*	- Time in seconds
in	*ts_nsec*	- Time in nanoseconds

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.13.3.5 int32 CFE_PSP_StartNTPDaemon (void)

Start the NTP client.

Description:

This function starts the NTP client task, ipntpd, on VxWorks.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

NTP client Task ID
CFE_PSP_ERROR

2.13.3.6 int32 CFE_PSP_StopNTPDaemon (void)

Stop the NTP client.

Description:

This function stops the NTP client task, ipntpd, on VxWorks.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.13.3.7 int32 CFE_PSP_Sync_From_OS_Enable (bool *enable*)

Enable/disable time sync.

Description:

This function sets the enabling/disabling of time sync. When the flag is true, the NTP Sync task actively trys to sync clocks. When the flag is false, the NTP Sync task will remain active without sync.

Assumptions, External Events, and Notes:

None

Parameters

in	*enable*	- Boolean flag for sync or not sync

Returns

True - If synchronized
False - If not synchronized

2.13.3.8 int32 CFE_PSP_Sync_From_OS_Freq (uint16 *new_frequency_sec*)

Change the sync frequency.

Description:

This function updates the NTP time synchronization frequency, in seconds.

Assumptions, External Events, and Notes:

If 0 is passed in, the function returns the current frequency.

Parameters

in	*new_frequency_sec*	- The new frequency, in seconds

Returns

CFE_PSP_SUCCESS - If successfully changed
Current frequency - If passed in 0 for new_frequency_sec

2.13.3.9 int32 CFE_PSP_TIME_Init (uint16 *timer_frequency_sec*)

Initialize the CFE PSP Time Task synchronizing with the NTP server.

Description:

This function intializes the cFE PSP Time sync task with the NTP server.

Assumptions, External Events, and Notes:

None

Parameters

in	*timer_frequency_sec*	- The update frequency, in seconds

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.13.3.10 void CFE_PSP_Update_OS_Time (void)

Update cFE time.

Description:

This function updates the time used by the cFE Time service.

Assumptions, External Events, and Notes:

This function will run forever until its task is deleted.

Parameters

None	

Returns

None

2.13.3.11 uint32 CFE_TIME_Micro2SubSecs (uint32)

Convert micro seconds in subseconds.

Description:

Defined in CFE module time cfe_time.h

2.13.3.12 void CFE_TIME_SetTime (CFE_TIME_SysTime_t)

Adjust CFE Time STCF so that local time match the new time.

Description:

Defined in CFE module time cfe_time_utils.h

2.13.3.13 int32 net_clock_vxworks_Destroy (void)

Gracefully shutdown NTP Sync Module.

Description:

Function will attempt to delete the task. Usually this function will be called when exiting cFS.

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.13.3.14 void ntp_clock_vxworks_Init (uint32 *PspModuleId*)

Entry point for the module.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

in,out	*PspModuleId*	- Unused

Returns

None

2.14 cfe_psp_sp0_info.c File Reference

API for collecting SP0(s) hardware and software information.

```
#include "cfe_psp.h"
#include <fcntl.h>
#include <stdio.h>
#include "ioLib.h"
#include <vxWorks.h>
#include "aimonUtil.h"
#include "sys950Lib.h"
#include "sysApi.h"
#include "scratchRegMap.h"
#include "bflashCt.h"
#include "tempSensor.h"
#include "cfe_psp_config.h"
#include "psp_sp0_info.h"
```

Functions

- int32 getSP0Info (void)

  Collect SP0 Hardware and Firmware data.
- void printSP0_info_table (void)

  Collect SP0 Hardware and Firmware data.
- void psp_dump_data (void)

  Function dumps the collected data to file.

2.14.1 Detailed Description

API for collecting SP0(s) hardware and software information.

Copyright

Description:

Functions here allow CFS to provide a method to probe SP0 hardware for information from POST, Temperatures, Voltages, Active Boot EEPROM, etc. In addition, this module has a function to save a dump_core text file before aborting CFS execution.

Limitations, Assumptions, External Events, and Notes:

None

2.14.2 Function Documentation

2.14.2.1 int32 getSP0Info (void)

Collect SP0 Hardware and Firmware data.

Description:

This function collects the SP0 hardware and firmware data and saves it in the sp0_info_table object, as well as a string in the sp0_data_dump object.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR - Never returns it

2.14.2.2 void printSP0_info_table (void)

Collect SP0 Hardware and Firmware data.

Description:

This function prints the SP0 data to the output console.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.14.2.3 void psp_dump_data (void)

Function dumps the collected data to file.

Description:

This function prints the SP0 data to the output console. Data is saved at SP0_DATA_DUMP_FILEPATH

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.15 cfe_psp_start.c File Reference

cFE PSP main entry point

```
#include <stdio.h>
#include <string.h>
#include <vxWorks.h>
#include <taskLib.h>
#include "target_config.h"
#include "scratchRegMap.h"
#include "aimonUtil.h"
#include "cfe_psp_config.h"
#include "common_types.h"
#include "osapi.h"
#include "cfe_psp.h"
#include "psp_start.h"
#include "cfe_psp_memory.h"
#include "cfe_psp_module.h"
#include "psp_mem_scrub.h"
#include "psp_sp0_info.h"
```

Macros

PSP Configuration

Description:

The preferred way to obtain the CFE tunable values at runtime is via the dynamically generated configuration object. This allows a single build of the PSP to be completely CFE-independent.

- #define CFE_PSP_MAIN_FUNCTION (∗GLOBAL_CONFIGDATA.CfeConfig->SystemMain)

  PSP Main function pointer.
- #define CFE_PSP_NONVOL_STARTUP_FILE (GLOBAL_CONFIGDATA.CfeConfig->NonvolStartupFile)

  PSP Non Volatile startup file.

Functions

- int OS_BSPMain (void)

  OSAL OS_BSPMain Entry Point.
- void CFE_PSP_Main (void)

  Main entry-point.
- void CFE_PSP_ProcessPOSTResults (void)

  Log the Power On Self Test (POST) results to the system log.
- static RESET_SRC_REG_ENUM CFE_PSP_ProcessResetType (void)

Determines the reset type and logs off nominal resets.

- void CFE_PSP_LogSoftwareResetType (RESET_SRC_REG_ENUM resetSrc)

  Determines if started in safe mode and logs off nominal resets.
- void OS_Application_Startup (void)

  Application startup entry point from OSAL BSP.
- void OS_Application_Run (void)

  Application Run entry point from OSAL BSP.
- int32 CFE_PSP_SuspendConsoleShellTask (bool suspend)

  Function Suspend/Resume the Console Shell Task.
- uint32 CFE_PSP_GetRestartType (uint32 *resetSubType)

  Get restart type.
- static int32 SetTaskPrio (const char *tName, int32 tgtPrio)

  Changes default task priority to a given priority.
- static int32 SetSysTasksPrio (void)

  Changes system task priorities so that they are lower than CFS system task priorities.
- unsigned int vxFpscrGet (void)

  Provides stub function for FPU exception handler, vxFpscrGet()
- void vxFpscrSet (unsigned int x)

  Provides stub function for FPU exception handler, vxFpscrSet()

Variables

- static uint32 ResetType = 0

  Reset Type.
- static uint32 ResetSubtype = 0

  Reset Sub Type.
- static USER_SAFE_MODE_DATA_STRUCT safeModeUserData

  Safe Mode User Data.
- static TASK_ID sg_uiShellTaskID = 0

  Console Shell Task ID.
- CFE_PSP_OS_Task_and_priority_t VxWorksTaskList []

  The list of VxWorks task to change the task priority to before finishing initialization.

2.15.1 Detailed Description

cFE PSP main entry point

Copyright

Description:

None

Limitations, Assumptions, External Events, and Notes:

None

2.15.2 Function Documentation

2.15.2.1 uint32 CFE_PSP_GetRestartType (uint32 * *resetSubType*)

Get restart type.

Description:

This function returns the last reset type. If a pointer to a valid memory space is passed in, it returns the reset sub-type in that memory. Right now the reset types are application-specific. For the cFE, they are defined in the cfe_es.h file.

Assumptions, External Events, and Notes:

None

Parameters

out	*resetSubType*	- Pointer to the variable that stores the returned reset sub-type

Returns

Last reset type

2.15.2.2 void CFE_PSP_LogSoftwareResetType (RESET_SRC_REG_ENUM *resetSrc*)

Determines if started in safe mode and logs off nominal resets.

Description:

None

Assumptions, External Events, and Notes:

RESET_SRC_REG_ENUM is defined in Aitech file scratchRegMap.h

Parameters

resetSrc	- Reset Type RESET_SRC_REG_ENUM

Returns

None

2.15.2.3 void CFE_PSP_Main (void)

Main entry-point.

Description:

This function is the entry point that the real time OS calls to start cFS. This function will do any BSP/OS-specific setup, then call the entry point of cFS, which is this function.

Assumptions, External Events, and Notes:

cFE should not call this function. See the description.

Parameters

None	

Returns

None

2.15.2.4 void CFE_PSP_ProcessPOSTResults (void)

Log the Power On Self Test (POST) results to the system log.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.15.2.5 static RESET_SRC_REG_ENUM CFE_PSP_ProcessResetType (void) [static]

Determines the reset type and logs off nominal resets.

Description:

Reset Types are defined in Aitech headers

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

RESET_SRC_POR
RESET_SRC_WDT
RESET_SRC_FWDT
RESET_SRC_CPCI
RESET_SRC_SWR

2.15.2.6 int32 CFE_PSP_SuspendConsoleShellTask (bool *suspend*)

Function Suspend/Resume the Console Shell Task.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

in	*suspend*	- True to suspend task, False to resume task

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.15.2.7 void OS_Application_Run (void)

Application Run entry point from OSAL BSP.

Description:

SP0 Implementation Specific

Assumptions, External Events, and Notes:

This function is declared but empty so that we don't run the default OSAL equivalent function. The latter will actively suspend the console shell.

Parameters

None	

Returns

None

2.15.2.8 void OS_Application_Startup (void)

Application startup entry point from OSAL BSP.

Description:

SP0 Implementation Specific

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.15.2.9 static int32 SetSysTasksPrio (void) [static]

Changes system task priorities so that they are lower than CFS system task priorities.

Description:

None

Assumptions, External Events, and Notes:

tNet0 priority should be adjusted to be right below what ever gets defined for CI/TO apps in your system if using the network interface CCSDS/UDP for CI/TO apps.

Parameters

None	

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.15.2.10 static int32 SetTaskPrio (const char * *tName*, int32 *tgtPrio*) [static]

Changes default task priority to a given priority.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

in	*tName*	- Task name
in	*tgtPrio*	- New task priority

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.15.2.11 unsigned int vxFpscrGet (void)

Provides stub function for FPU exception handler, vxFpscrGet()

Description:

Added this function here so that the code can compile & run without error.

If there's code that calls these functions, we will get a message like so, > ld < cfe-core.o Warning: module 0x461d010 holds reference to undefined symbol vxFpscrGet. Warning: module 0x461d010 holds reference to undefined symbol vxFpscrSet.

These do not seem to be included in 85xx build, but are defined as "defined(_PPC_) && CPU != PPC440" in vxWorks osapi.c, line 2707, v4.2.1a

If this function is not used, stub it out like below. Otherwise, define it.

Assumptions, External Events, and Notes:

If still relevant, have OSAL add conditional compile when SPE preset instead of FPU Once that has occurred we can remove vxFpscrGet and vxFpscrSet

Parameters

None	

Returns

0 - Integer Zero

2.15.2.12 void vxFpscrSet (unsigned int *x*)

Provides stub function for FPU exception handler, vxFpscrSet()

Description:

Added this function here so that the code can compile & run without error.

If there's code that calls these functions, we will get a message like so, > ld < cfe-core.o Warning: module 0x461d010 holds reference to undefined symbol vxFpscrGet. Warning: module 0x461d010 holds reference to undefined symbol vxFpscrSet.

These do not seem to be included in 85xx build, but are defined as "defined(_PPC_) && CPU != PPC440" in vxWorks osapi.c, line 2707, v4.2.1a

If this function is not used, stub it out like below. Otherwise, define it.

Assumptions, External Events, and Notes:

If still relevant, have OSAL add conditional compile when SPE preset instead of FPU Once that has occurred we can remove vxFpscrGet and vxFpscrSet

Parameters

x	- Unused

Returns

None

2.15.3 Variable Documentation

2.15.3.1 CFE_PSP_OS_Task_and_priority_t VxWorksTaskList[ ]

Initial value:

```
=
{
    {"tLogTask", 0},
    {"tShell0", 201},
    {"tWdbTask", 203},
    {"tVxdbgTask", 200},
    {"tNet0", 25},
    {"ipftps", 202},
    {"ipcom_syslogd", 205},
    {"ipcom_telnetd", 204}
}
```

The list of VxWorks task to change the task priority to before finishing initialization.

2.16 cfe_psp_support.c File Reference

Contains glue routines between the cFE and the OS Board Support Package (BSP)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "vxWorks.h"
#include "cacheLib.h"
#include "rebootLib.h"
#include "common_types.h"
#include "osapi.h"
#include "cfe_psp.h"
#include "cfe_psp_memory.h"
#include "psp_sp0_info.h"
#include "target_config.h"
```

Macros

Macros

- #define CFE_PSP_CPU_ID (GLOBAL_CONFIGDATA.Default_CpuId)
  CPU ID.
- #define CFE_PSP_CPU_NAME (GLOBAL_CONFIGDATA.Default_CpuName)
  CPU NAME.
- #define CFE_PSP_SPACECRAFT_ID (GLOBAL_CONFIGDATA.Default_SpacecraftId)
  SPACECRAFT ID.

Functions

- void CFE_PSP_Restart (uint32 resetType)
  Re-start.
- void CFE_PSP_Panic (int32 errorCode)
  Abort cFE startup.
- void CFE_PSP_FlushCaches (uint32 type, void ∗address, uint32 size)
  Flush memory caches.
- uint32 CFE_PSP_GetProcessorId (void)
  Get the CPU ID.
- uint32 CFE_PSP_GetSpacecraftId (void)
  Get the spacecraft ID.
- const char ∗ CFE_PSP_GetProcessorName (void)
  Get the processor name.

Variables

- CFE_PSP_MemoryBlock_t PSP_ReservedMemBlock
  Pointer to the reserved memory block.

2.16.1 Detailed Description

Contains glue routines between the cFE and the OS Board Support Package (BSP)

Copyright

Description:

The functions here allow the cFE to interface functions that are board and OS specific and usually dont fit well in the OS abstraction layer.

Limitations, Assumptions, External Events, and Notes:

None

2.16.2 Function Documentation

2.16.2.1 void CFE_PSP_FlushCaches (uint32 *type*, void ∗ *address*, uint32 *size*)

Flush memory caches.

Description:

This function flushes the processor caches. This function is in the PSP because it is sometimes implemented in hardware and sometimes taken care of by the OS.

Assumptions, External Events, and Notes:

This function is not implemented for the SP0-vxworks6.9 PSP since it is managed by the SP0 BSP/VxWorks OS.

Parameters

in	*type*	- Cache memory type
in	*address*	- Pointer to the cache memory address
in	*size*	- Cache memory size

Returns

None

2.16.2.2 uint32 CFE_PSP_GetProcessorId (void)

Get the CPU ID.

Description:

This function returns the CPU ID as pre-fdefined by the cFE for specific board and BSP.

Assumptions, External Events, and Notes:

The macro is defined in cfe_platform_cfg.h.

Parameters

None	

Returns

CFE_PSP_CPU_ID

2.16.2.3 const char* CFE_PSP_GetProcessorName (void)

Get the processor name.

Description:

This function returns the CPU name as pre-defined by the cFE.

Assumptions, External Events, and Notes:

The macro is defined in cfe_platform_cfg.h.

Parameters

None	

Returns

CFE_PSP_CPU_NAME

2.16.2.4 uint32 CFE_PSP_GetSpacecraftId (void)

Get the spacecraft ID.

Description:

This function returns the spacecraft ID as pre-defined by the cFE.

Assumptions, External Events, and Notes:

The macro is defined in cfe_platform_cfg.h.

Parameters

None	

Returns

CFE_PSP_SPACECRAFT_ID

2.16.2.5 void CFE_PSP_Panic (int32 *errorCode*)

Abort cFE startup.

Description:

This function provides the mechanism to abort the cFE startup process and returns back to the OS.

Assumptions, External Events, and Notes:

This function should not be called by the cFS applications.

Parameters

in	*errorCode*	- Error code that causes the exit

Returns

None

2.16.2.6 void CFE_PSP_Restart (uint32 *resetType*)

Re-start.

Description:

This function is the entry point back to the BSP to restart the processor. cFE calls this function to restart the processor.

Assumptions, External Events, and Notes:

None

Parameters

in	*resetType*	- Type of cFE reset

Returns

None

2.17 cfe_psp_version.c File Reference

Defines API that obtains the values of the various version identifiers.

```
#include "cfe_psp.h"
#include "psp_version.h"
```

Functions

- const char * CFE_PSP_GetVersionString (void)

  Obtain the PSP version/baseline identifier string.
- const char * CFE_PSP_GetVersionCodeName (void)

  Obtain the version code name.
- void CFE_PSP_GetVersionNumber (uint8 VersionNumbers[4])

  Obtain the PSP numeric version numbers as uint8 values.
- uint32 CFE_PSP_GetBuildNumber (void)

  Obtain the PSP library numeric build number.

2.17.1 Detailed Description

Defines API that obtains the values of the various version identifiers.

Description:

GSC-18128-1, "Core Flight Executive Version 6.7"

2.17.2 Function Documentation

2.17.2.1 uint32 CFE_PSP_GetBuildNumber (void)

Obtain the PSP library numeric build number.

Description:

The build number is a monotonically increasing number that (coarsely) reflects the number of commits/changes that have been merged since the epoch release. During development cycles this number should increase after each subsequent merge/modification.
Like other version information, this is a fixed number assigned at compile time.

Assumptions, External Events, and Notes:

None

Returns

The OSAL library build number

2.17.2.2 const char* CFE_PSP_GetVersionCodeName (void)

Obtain the version code name.

Description:

This retrieves the PSP code name.
This is a compatibility indicator for the overall NASA CFS ecosystem.
All modular components which are intended to interoperate should report the same code name.

Assumptions, External Events, and Notes:

None

Returns

Code name. This is a fixed string and cannot be NULL.

2.17.2.3 void CFE_PSP_GetVersionNumber (uint8 *VersionNumbers[4]*)

Obtain the PSP numeric version numbers as uint8 values.

Description:

This retrieves the numeric PSP version identifier as an array of 4 uint8 values.
The array of numeric values is in order of precedence: [0] = Major Number [1] = Minor Number [2] = Revision Number [3] = Mission Revision
The "Mission Revision" (last output) also indicates whether this is an official release, a patched release, or a development version. 0 indicates an official release 1-254 local patch level (reserved for mission use) 255 indicates a development build

Assumptions, External Events, and Notes:

None

Parameters

out	*VersionNumbers*	A fixed-size array to be filled with the version numbers

Returns

None

2.17.2.4 const char* CFE_PSP_GetVersionString (void)

Obtain the PSP version/baseline identifier string.

Description:

This retrieves the PSP version identifier string without extra info.

Assumptions, External Events, and Notes:

None

Returns

Version string. This is a fixed string and cannot be NULL.

2.18 cfe_psp_watchdog.c File Reference

API to support Watchdog.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "common_types.h"
#include "osapi.h"
#include "cfe_psp.h"
#include "sysApi.h"
```

Functions

- void CFE_PSP_WatchdogInit (void)

  Initialize the watchdog timer.
- void CFE_PSP_WatchdogEnable (void)

  Enable the watchdog timer.
- void CFE_PSP_WatchdogDisable (void)

  Disable the watchdog timer.
- void CFE_PSP_WatchdogService (void)

  Service the watchdog timer.
- uint32 CFE_PSP_WatchdogGet (void)

  Get the watchdog time.
- void CFE_PSP_WatchdogSet (uint32 watchDogValue)

  Set the watchdog time.

Variables

- uint32 CFE_PSP_WatchdogValue = 20000

  Watchdog current millisecond value.

2.18.1 Detailed Description

API to support Watchdog.

Copyright

Description:

API to enable/disable, and control watchdog

Limitations, Assumptions, External Events, and Notes:

None

2.18.2 Function Documentation

2.18.2.1 void CFE_PSP_WatchdogDisable (void)

Disable the watchdog timer.

Description:

This function disables the watchdog timer.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.18.2.2 void CFE_PSP_WatchdogEnable (void)

Enable the watchdog timer.

Description:

This function enables the watchdog timer.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.18.2.3 uint32 CFE_PSP_WatchdogGet (void)

Get the watchdog time.

Description:

This function fetches the watchdog time, in milliseconds.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

The watchdog time in milliseconds

2.18.2.4 void CFE_PSP_WatchdogInit (void)

Initialize the watchdog timer.

Description:

This function configures and intializes the watchdog timer.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.18.2.5 void CFE_PSP_WatchdogService (void)

Service the watchdog timer.

Description:

This function services the watchdog timer according to the value set in CFE_PSP_WatchdogSet().

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.18.2.6 void CFE_PSP_WatchdogSet (uint32 *watchDogValue*)

Set the watchdog time.

Description:

This function sets the current watchdog time, in milliseconds.

Assumptions, External Events, and Notes:

None

Parameters

in	*watchDogValue*	- watchdog time in milliseconds

Returns

None

2.19 psp_cds_flash.h File Reference

API to save and restore CDS in FLASH memory.

Functions

- void CFE_PSP_SetReadCDSMethod (uint8 ucMethod)

  Set the CDS reading method.
- uint8 CFE_PSP_GetReadCDSMethod ()

  Get the CDS reading method.
- void CFE_PSP_SetStaticCRC (uint32 uiNewCRC)

  Change the previous calculated CRC value to new provided value.
- uint32 CFE_PSP_GetStaticCRC (void)

  Get the previous calculated CRC value.
- uint32 CFE_PSP_CalculateCRC (const void ∗DataPtr, uint32 DataLength, uint32 InputCRC)

  Calculate 16 bits CRC from input data.
- int32 CFE_PSP_ReadCDSFromFlash (uint32 ∗puiReadBytes)

  Read the whole CDS data from Flash.
- int32 CFE_PSP_WriteCDSToFlash (uint32 ∗puiWroteBytes)

  Write the whole CDS data on Flash.

2.19.1 Detailed Description

API to save and restore CDS in FLASH memory.

Copyright

Description:

None

Limitations, Assumptions, External Events, and Notes:

None

2.19.2 Function Documentation

2.19.2.1 uint32 CFE_PSP_CalculateCRC (const void * *DataPtr*, uint32 *DataLength*, uint32 *InputCRC*)

Calculate 16 bits CRC from input data.

Description:

None

Assumptions, External Events, and Notes:

InputCRC allows the user to calculate the CRC of non-contiguous blocks as a single value. Nominally, the user should set this value to zero.

CFE now includes a function to calculate the CRC. uint32 CFE_ES_CalculateCRC(void *pData, uint32 DataLength, uint32 InputCRC, uint32 TypeCRC); Only CFE_MISSION_ES_CRC_16 is implemented as the TypeCRC

Parameters

in,out	*DataPtr*	- Pointer to the input data buffer
in,out	*DataLength*	- Data buffer length
in,out	*InputCRC*	- A starting value for use in the CRC calculation.

Returns

Calculated CRC value

2.19.2.2 uint8 CFE_PSP_GetReadCDSMethod ( )

Get the CDS reading method.

Description:

This function get the CDS reading method(use CRC, always read from Flash, or trust the CDS reserved memory in RAM is correct.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

CFE_PSP_CDS_READ_METHOD_DEFAULT - Trust the CDS data on RAM (no CRC or read from Flash)
CFE_PSP_CDS_READ_METHOD_CRC - Check the CRC first then read from Flash if CRC mis-matched
CFE_PSP_CDS_READ_METHOD_FLASH - Always read from Flash

2.19.2.3 uint32 CFE_PSP_GetStaticCRC (void)

Get the previous calculated CRC value.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

Calculated CRC value

2.19.2.4 int32 CFE_PSP_ReadCDSFromFlash (uint32 * *puiReadBytes*)

Read the whole CDS data from Flash.

Description:

This function read the whole CDS data on Flash to reserved memory on RAM.

Warning

It took about 117ms to read 131072 bytes (128KB) whole CDS area from Flash.

Assumptions, External Events, and Notes:

None

Parameters

puiReadBytes	- Number of read bytes

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.19.2.5 void CFE_PSP_SetReadCDSMethod (uint8 *ucMethod*)

Set the CDS reading method.

Description:

This function set the CDS reading method(use CRC, always read from Flash, or trust the CDS reserved memory in RAM is correct.

Assumptions, External Events, and Notes:

None

Parameters

in	*ucMethod*	- Reading method

Returns

None

2.19.2.6 void CFE_PSP_SetStaticCRC (uint32 *uiNewCRC*)

Change the previous calculated CRC value to new provided value.

Description:

This function change the previous calculated CRC value to new provided value. This function is just for testing purpose by forcing the CRC mismatched and read CDS data from Flash.

Assumptions, External Events, and Notes:

None

Parameters

uiNewCRC	- New CRC

Returns

None

2.19.2.7 int32 CFE_PSP_WriteCDSToFlash (uint32 * *puiWroteBytes*)

Write the whole CDS data on Flash.

Description:

This function write the whole CDS data from reserved memory on RAM to Flash.

Assumptions, External Events, and Notes:

It took about 117ms to write 131072 bytes (128KB) whole CDS data to Flash.

Parameters

puiWroteBytes	- Number of written bytes

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.20 psp_mem_scrub.h File Reference

API to control Memory Scrubbing.

Functions

- void CFE_PSP_MEM_SCRUB_Set (uint32 newStartAddr, uint32 newEndAddr, osal_priority_t task_priority)

  Set the Memory Scrubbing parameters.
- bool CFE_PSP_MEM_SCRUB_isRunning (void)

  Check if the Memory Scrubbing task is running.
- void CFE_PSP_MEM_SCRUB_Delete (void)

  Stop the memory scrubbing task.
- void CFE_PSP_MEM_SCRUB_Status (void)

  Print the Memory Scrubbing statistics.
- void CFE_PSP_MEM_SCRUB_Task (void)

  Memory Scrubbing task.
- void CFE_PSP_MEM_SCRUB_Init (void)

  Initialize the Memory Scrubbing task.
- void CFE_PSP_MEM_SCRUB_Enable (void)

  Enable the Memory Scrubbing task.
- void CFE_PSP_MEM_SCRUB_Disable (void)

  Disable the Memory Scrubbing task.

2.20.1 Detailed Description

API to control Memory Scrubbing.

Copyright

Description

This file contains the function prototypes relating to memory scrubbing. This is specific to the SP0-S processor running VxWorks 6.9 OS.

Limitations, Assumptions, External Events, and Notes:

None

2.20.2 Function Documentation

2.20.2.1 void CFE_PSP_MEM_SCRUB_Delete (void)

Stop the memory scrubbing task.

Description:

This function deletes the Memory Scrubbing task. The task is deleted and the statistics are reset.

Assumptions, External Events, and Notes:

None

Parameters

-	None

Returns

None

2.20.2.2 void CFE_PSP_MEM_SCRUB_Disable (void)

Disable the Memory Scrubbing task.

Description:

This function disables the Memory Scrubbing task.

Assumptions, External Events, and Notes:

If the task is already running, delete it. If the task is not running, then do nothing.

Parameters

None	

Returns

None

2.20.2.3 void CFE_PSP_MEM_SCRUB_Enable (void)

Enable the Memory Scrubbing task.

Description:

This function enables the Memory Scrubbing task.

Assumptions, External Events, and Notes:

If the task is already running, do nothing. If the task is not running, then start it.

Parameters

None	

Returns

None

2.20.2.4 void CFE_PSP_MEM_SCRUB_Init (void)

Initialize the Memory Scrubbing task.

Description:

This function starts the Memory Scrubbing task as a child thread.

Assumptions, External Events, and Notes:

The scrubMemory function implemented by AiTech may never return an error.

Parameters

None	

Returns

None

2.20.2.5 bool CFE_PSP_MEM_SCRUB_isRunning (void)

Check if the Memory Scrubbing task is running.

Description:

This function provides the status whether the Memory Scrubbing task is running.

Assumptions, External Events, and Notes:

None

Parameters

-	None

Returns

true - If task is running
false - If task is not running

2.20.2.6 void CFE_PSP_MEM_SCRUB_Set (uint32 *newStartAddr*, uint32 *newEndAddr*, osal_priority_t *task_priority*)

Set the Memory Scrubbing parameters.

Description:

This functions set the memory scrubbing parameters.

Assumptions, External Events, and Notes:

After calling this function, the new settings will be applied in the next call to the Activate Memory Scrubbing funtion. If newEndAddr is set to a value larger than the actual physical memory limit, the function will use the phyisical memory limit. Task priority can only be set between MEMSCRUB_PRIORITY_UP_RANGE and MEMSCRUB_PRIORITY_DOWN_RANGE defined in cfe_psp_config.h. Default is set to MEMSCRUB_DEFAULT_PRIORITY.

Parameters

in	*newStartAddr*	- Memory address to start from, usually zero
in	*newEndAddr*	- Memory address to end at, usually end of the physical RAM
in	*task_priority*	- The task priority

Returns

None

2.20.2.7 void CFE_PSP_MEM_SCRUB_Status (void)

Print the Memory Scrubbing statistics.

Description:

This function outputs to the console the following Memory Scrubbing statistics: Start memory address, End memory address, current memory page and total memory pages

Assumptions, External Events, and Notes:

Start memory address is usually 0. End memory address is usually set to the last value of RAM address. Note that a page is 4098 bytes.

Parameters

None	

Returns

None

2.20.2.8 void CFE_PSP_MEM_SCRUB_Task (void)

Memory Scrubbing task.

Description:

This function performs the Memory Scrubbing steps.

Assumptions, External Events, and Notes:

The scrubMemory function implemented by AiTech may never return an error.

Parameters

None	

Returns

None

Memory Scrubbing task.

Description:

This is the main function for the Memory Scrubbing task.

Assumptions, External Events, and Notes:

The scrubMemory function implemented by AiTech may never return an error.

Parameters

None	

Returns

None

2.21 psp_sp0_info.h File Reference

API to collect and dump SP0 Hardware/Software information.

Macros

- #define SP0_TEXT_BUFFER_MAX_SIZE 1000
  SP0_TEXT_BUFFER_MAX_SIZE.

Variables

SP0 Information String Buffer

- char sp0_data_dump [SP0_TEXT_BUFFER_MAX_SIZE]
  SP0 String Buffer.
- int sp0_data_dump_length
  Actual length of the string buffer.

SP0 info structure

Description:

The table includes values that changes only once during boot and others that changes at a regular interval.

Variables that changes at regular intervals are:

- systemStartupUsecTime
- temperatures
- voltages

- struct {
  char * systemModel
  Pointer to the string identifing the System Model.
  char * systemBspRev
  Pointer to the string identifing the system BSP Revision.
  uint32 systemPhysMemTop
  Top of the System Physical Memmory.
  int systemProcNum
  Number of Processors.
  int systemSlotId
  Slod ID in the chassis.
  bool systemCpciSysCtrl

Identifies if the SP0 is the cPCI main system controller.
uint32 systemCoreClockSpeed
System Core Clock Speed in MHz.
uint8 systemLastResetReason
Reason for last SP0 computer reset.
uint8 active_boot
Identifies the EEPROM to successfully booted the kernel.
int systemClkRateGet
System Clock Rate.
int systemAuxClkRateGet
System Aux Clock Rate.
uint64 bitExecuted
Identifies the POST Test Bit Executed.
uint64 bitResult
Identifies the POST Test Results.
char safeModeUserData [256]
Safe Mode User Data.
double systemStartupUsecTime
Number of usec since startup.
float temperatures [4]
Array of 4 temperatures on the SP0 computer.
float voltages [6]
Array of 6 voltages powering the SP0.
} **sp0_info_table**

- int32 getSP0Info (void)

  Collect SP0 Hardware and Firmware data.
- void printSP0_info_table (void)

  Collect SP0 Hardware and Firmware data.
- void psp_dump_data (void)

  Function dumps the collected data to file.

2.21.1 Detailed Description

API to collect and dump SP0 Hardware/Software information.

Copyright

This software was created at NASA's Johnson Space Center. This software is governed by the NASA Open Source Agreement and may be used, distributed and modified only pursuant to the terms of that agreement.

Description

This file contains the function prototypes to access the SP0 hardware and software data. It also dumps the same data to FLASH memory in the case when cFS calls Panic.

Limitations, Assumptions, External Events, and Notes:

None

2.21.2 Macro Definition Documentation

2.21.2.1 #define SP0_TEXT_BUFFER_MAX_SIZE 1000

SP0_TEXT_BUFFER_MAX_SIZE.

Description:

This is the maximum size of the SP0 char array table.

2.21.3 Function Documentation

2.21.3.1 int32 getSP0Info (void)

Collect SP0 Hardware and Firmware data.

Description:

This function collects the SP0 hardware and firmware data and saves it in the sp0_info_table object, as well as a string in the sp0_data_dump object.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR - Never returns it

2.21.3.2 void printSP0_info_table (void)

Collect SP0 Hardware and Firmware data.

Description:

This function prints the SP0 data to the output console.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.21.3.3 void psp_dump_data (void)

Function dumps the collected data to file.

Description:

This function prints the SP0 data to the output console. Data is saved at SP0_DATA_DUMP_FILEPATH

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.22 psp_start.h File Reference

Header file for the PSP function prototypes in cfe_psp_start.c.

Functions

- void CFE_PSP_ProcessPOSTResults (void)
  Log the Power On Self Test (POST) results to the system log.
- static RESET_SRC_REG_ENUM CFE_PSP_ProcessResetType (void)
  Determines the reset type and logs off nominal resets.
- void CFE_PSP_LogSoftwareResetType (RESET_SRC_REG_ENUM resetSrc)
  Determines if started in safe mode and logs off nominal resets.
- void OS_Application_Startup (void)
  Application startup entry point from OSAL BSP.
- void OS_Application_Run (void)
  Application Run entry point from OSAL BSP.
- int32 CFE_PSP_SuspendConsoleShellTask (bool suspend)
  Function Suspend/Resume the Console Shell Task.
- uint32 CFE_PSP_GetRestartType (uint32 *resetSubType)
  Get restart type.
- static int32 SetTaskPrio (const char *tName, int32 tgtPrio)
  Changes default task priority to a given priority.
- static int32 SetSysTasksPrio (void)
  Changes system task priorities so that they are lower than CFS system task priorities.
- unsigned int vxFpscrGet (void)
  Provides stub function for FPU exception handler, vxFpscrGet()
- void vxFpscrSet (unsigned int x)
  Provides stub function for FPU exception handler, vxFpscrSet()

2.22.1 Detailed Description

Header file for the PSP function prototypes in cfe_psp_start.c.

Copyright

This software was created at NASA's Johnson Space Center. This software is governed by the NASA Open Source Agreement and may be used, distributed and modified only pursuant to the terms of that agreement.

Description:

None

Limitations, Assumptions, External Events, and Notes:

None

2.22.2 Function Documentation

2.22.2.1 uint32 CFE_PSP_GetRestartType (uint32 * *resetSubType*)

Get restart type.

Description:

This function returns the last reset type. If a pointer to a valid memory space is passed in, it returns the reset sub-type in that memory. Right now the reset types are application-specific. For the cFE, they are defined in the cfe_es.h file.

Assumptions, External Events, and Notes:

None

Parameters

out	*resetSubType*	- Pointer to the variable that stores the returned reset sub-type

Returns

Last reset type

2.22.2.2 void CFE_PSP_LogSoftwareResetType (RESET_SRC_REG_ENUM *resetSrc*)

Determines if started in safe mode and logs off nominal resets.

Description:

None

Assumptions, External Events, and Notes:

RESET_SRC_REG_ENUM is defined in Aitech file scratchRegMap.h

Parameters

resetSrc	- Reset Type RESET_SRC_REG_ENUM

Returns

None

2.22.2.3 void CFE_PSP_ProcessPOSTResults (void)

Log the Power On Self Test (POST) results to the system log.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.22.2.4 static RESET_SRC_REG_ENUM CFE_PSP_ProcessResetType (void) [static]

Determines the reset type and logs off nominal resets.

Description:

None

Assumptions, External Events, and Notes:

Output defines are defined in Aitech file scratchRegMap.h

Parameters

None	

Returns

RESET_SRC_POR
RESET_SRC_WDT
RESET_SRC_FWDT
RESET_SRC_CPCI
RESET_SRC_SWR

2.22.2.5 int32 CFE_PSP_SuspendConsoleShellTask (bool *suspend*)

Function Suspend/Resume the Console Shell Task.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

in	*suspend*	- True to suspend task, False to resume task

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.22.2.6 void OS_Application_Run (void)

Application Run entry point from OSAL BSP.

Description:

SP0 Implementation Specific

Assumptions, External Events, and Notes:

This function is declared but empty so that we don't run the default OSAL equivalent function. The latter will actively suspend the console shell.

Parameters

None	

Returns

None

2.22.2.7 void OS_Application_Startup (void)

Application startup entry point from OSAL BSP.

Description:

SP0 Implementation Specific

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

None

2.22.2.8 static int32 SetSysTasksPrio (void) [static]

Changes system task priorities so that they are lower than CFS system task priorities.

Description:

None

Assumptions, External Events, and Notes:

tNet0 priority should be adjusted to be right below what ever gets defined for CI/TO apps in your system if using the network interface CCSDS/UDP for CI/TO apps.

Parameters

None	

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.22.2.9 static int32 SetTaskPrio (const char * *tName*, int32 *tgtPrio*) [static]

Changes default task priority to a given priority.

Description:

None

Assumptions, External Events, and Notes:

None

Parameters

in	*tName*	- Task name
in	*tgtPrio*	- New task priority

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.22.2.10 unsigned int vxFpscrGet (void)

Provides stub function for FPU exception handler, vxFpscrGet()

Description:

Added this function here so that the code can compile & run without error.

If there's code that calls these functions, we will get a message like so, > ld < cfe-core.o Warning: module 0x461d010 holds reference to undefined symbol vxFpscrGet. Warning: module 0x461d010 holds reference to undefined symbol vxFpscrSet.

These do not seem to be included in 85xx build, but are defined as "defined(_PPC_) && CPU != PPC440" in vxWorks osapi.c, line 2707, v4.2.1a

If this function is not used, stub it out like below. Otherwise, define it.

Assumptions, External Events, and Notes:

If still relevant, have OSAL add conditional compile when SPE preset instead of FPU Once that has occurred we can remove vxFpscrGet and vxFpscrSet

Parameters

None	

Returns

0 - Integer Zero

2.22.2.11 void vxFpscrSet (unsigned int *x*)

Provides stub function for FPU exception handler, vxFpscrSet()

Description:

Added this function here so that the code can compile & run without error.

If there's code that calls these functions, we will get a message like so, > ld < cfe-core.o Warning: module 0x461d010 holds reference to undefined symbol vxFpscrGet. Warning: module 0x461d010 holds reference to undefined symbol vxFpscrSet.

These do not seem to be included in 85xx build, but are defined as "defined(_PPC_) && CPU != PPC440" in vxWorks osapi.c, line 2707, v4.2.1a

If this function is not used, stub it out like below. Otherwise, define it.

Assumptions, External Events, and Notes:

If still relevant, have OSAL add conditional compile when SPE preset instead of FPU Once that has occurred we can remove vxFpscrGet and vxFpscrSet

Parameters

x	- Unused

Returns

None

2.23 psp_time_sync.h File Reference

Header API to control NTP Sync.

Functions

- int32 CFE_PSP_TIME_Init (uint16 timer_frequency_sec)
  Initialize the CFE PSP Time Task synchronizing with the NTP server.
- int32 CFE_PSP_Sync_From_OS_Enable (bool enable)
  Enable/disable time sync.
- bool CFE_PSP_NTP_Daemon_Get_Status (void)
  Get the NTP daemon status.
- int32 net_clock_vxworks_Destroy (void)
  Gracefully shutdown NTP Sync Module.
- int32 CFE_PSP_Sync_From_OS_Freq (uint16 new_frequency_sec)
  Change the sync frequency.
- int32 CFE_PSP_Set_OS_Time (const uint32 ts_sec, const uint32 ts_nsec)
  Set the OS time.
- int32 CFE_PSP_Get_OS_Time (CFE_TIME_SysTime_t ∗myT)
  Gets the current time from VxWorks OS.
- bool CFE_PSP_TimeService_Ready (void)
  Check if CFS Time Service is up and running.
- void CFE_PSP_Update_OS_Time (void)
  Update cFE time.
- int32 CFE_PSP_StartNTPDaemon (void)
  Start the NTP client.
- int32 CFE_PSP_StopNTPDaemon (void)
  Stop the NTP client.
- int32 CFE_PSP_NTP_Daemon_Enable (bool enable)
  Enable/disable the NTP client.

2.23.1 Detailed Description

Header API to control NTP Sync.

Copyright

Description:

This file contains the function prototypes that synchronize the cFE Time services to the NTP server. Note that the NTP server must be built into the kernel.

Limitations, Assumptions, External Events, and Notes:

The way this module updates the local time is by calling the CFE Time Service function CFE_TIME_SetTime(). The function changes the STCF value.
GSFC developers do not recommend to use this method of updating CFE time, but rather to use the function CFE-_TIME_ExternalTime(). The only way to use this function is by building an app that will periodically (1Hz) get NTP time and publish it via Software Bus.

2.23.2 Function Documentation

2.23.2.1 int32 CFE_PSP_Get_OS_Time (CFE_TIME_SysTime_t * *myT*)

Gets the current time from VxWorks OS.

Description:

This function gets the current VxWorks OS time.

Assumptions, External Events, and Notes:

This function is used by the NTP Sync task to grab the current OS time. It uses CLOCK_REALTIME.

Parameters

in,out	*myT*	- Pointer to the variable that stores the returned time value

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.23.2.2 int32 CFE_PSP_NTP_Daemon_Enable (bool *enable*)

Enable/disable the NTP client.

Description:

This function enables/disables the NTP client task, ipntpd, on VxWorks.

Assumptions, External Events, and Notes:

None

Parameters

in	*enable*	- Boolean flag for enable or disable

Returns

NTP client task ID - If successfully starts the NTP clien task
CFE_PSP_SUCCESS - If successfully stops the NTP client task
CFE_PSP_ERROR

2.23.2.3 bool CFE_PSP_NTP_Daemon_Get_Status (void)

Get the NTP daemon status.

Description:

This function checks if the VxWorks NTP client task is running. It does not check if the task has successfully synchronized with an NTP server.

Assumptions, External Events, and Notes:

The task name for the VxWorks NTP client is the default "ipntpd".

Parameters

None	

Returns

True - If NTP client task is running
False - If NTP client task is not running

2.23.2.4 int32 CFE_PSP_Set_OS_Time (const uint32 *ts_sec*, const uint32 *ts_nsec*)

Set the OS time.

Description:

This function sets the VxWorks OS time.

Assumptions, External Events, and Notes:

The changes do not occur if the NTP client is setup to synchrone with an NTP server. Set the OS CLOCK_REALTIME to a specified timestamp. Parameters are in UNIX time format, since Epoch 1/1/1970.

Parameters

in	*ts_sec*	- Time in seconds
in	*ts_nsec*	- Time in nanoseconds

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.23.2.5 int32 CFE_PSP_StartNTPDaemon (void)

Start the NTP client.

Description:

This function starts the NTP client task, ipntpd, on VxWorks.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

NTP client Task ID
CFE_PSP_ERROR

2.23.2.6 int32 CFE_PSP_StopNTPDaemon (void)

Stop the NTP client.

Description:

This function stops the NTP client task, ipntpd, on VxWorks.

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.23.2.7 int32 CFE_PSP_Sync_From_OS_Enable (bool *enable*)

Enable/disable time sync.

Description:

This function sets the enabling/disabling of time sync. When the flag is true, the NTP Sync task actively trys to sync clocks. When the flag is false, the NTP Sync task will remain active without sync.

Assumptions, External Events, and Notes:

None

Parameters

in	*enable*	- Boolean flag for sync or not sync

Returns

True - If synchronized
False - If not synchronized

2.23.2.8 int32 CFE_PSP_Sync_From_OS_Freq (uint16 *new_frequency_sec*)

Change the sync frequency.

Description:

This function updates the NTP time synchronization frequency, in seconds.

Assumptions, External Events, and Notes:

If 0 is passed in, the function returns the current frequency.

Parameters

in	*new_frequency_sec*	- The new frequency, in seconds

Returns

CFE_PSP_SUCCESS - If successfully changed
Current frequency - If passed in 0 for new_frequency_sec

2.23.2.9 int32 CFE_PSP_TIME_Init (uint16 *timer_frequency_sec*)

Initialize the CFE PSP Time Task synchronizing with the NTP server.

Description:

This function intializes the cFE PSP Time sync task with the NTP server.

Assumptions, External Events, and Notes:

None

Parameters

in	*timer_frequency_sec*	- The update frequency, in seconds

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.23.2.10 bool CFE_PSP_TimeService_Ready (void)

Check if CFS Time Service is up and running.

Description:

Assumptions, External Events, and Notes:

None

Parameters

None	

Returns

true - CFE Time Service is ready
false - CFE Time Service is not ready

2.23.2.11 void CFE_PSP_Update_OS_Time (void)

Update cFE time.

Description:

This function updates the time used by the cFE Time service.

Assumptions, External Events, and Notes:

This function will run forever until its task is deleted.

Parameters

None	

Returns

None

2.23.2.12 int32 net_clock_vxworks_Destroy (void)

Gracefully shutdown NTP Sync Module.

Description:

Function will attempt to delete the task. Usually this function will be called when exiting cFS.

Returns

CFE_PSP_SUCCESS
CFE_PSP_ERROR

2.24 psp_version.h File Reference

Defines API that obtains the values of the various version identifiers.

Macros

Version Macro Definitions

- #define CFE_PSP_IMPL_BUILD_NUMBER 112
  Development Build Macro Definitions - Build Number.
- #define CFE_PSP_IMPL_BUILD_BASELINE "v1.5.0-rc1"
  Development Build Macro Definitions - Baseline.
- #define CFE_PSP_IMPL_MAJOR_VERSION 1
  ONLY APPLY for OFFICIAL releases. Major version number.
- #define CFE_PSP_IMPL_MINOR_VERSION 4
  ONLY APPLY for OFFICIAL releases. Minor version number.
- #define CFE_PSP_IMPL_REVISION 0
  ONLY APPLY for OFFICIAL releases. Revision number.
- #define CFE_PSP_IMPL_MISSION_REV 99
  ONLY APPLY for OFFICIAL releases. Revision version number. A value of "99" indicates an unreleased development version.
- #define CFE_PSP_IMPL_CODENAME "Bootes"
  ONLY APPLY for OFFICIAL releases. Codename.

Tools to construct version string

- #define CFE_PSP_IMPL_STR_HELPER(x) #x
  Helper function to concatenate strings from integer.
- #define CFE_PSP_IMPL_STR(x) CFE_PSP_IMPL_STR_HELPER(x)
  Helper function to concatenate strings from integer.
- #define CFE_PSP_IMPL_VERSION CFE_PSP_IMPL_BUILD_BASELINE "+dev" CFE_PSP_IMPL_STR(CFE_PSP_IMPL_BUILD_NUMBER)
  DEVELOPMENT Build Version Number.
- #define CFE_PSP_IMPL_VERSION_STRING
  DEVELOPMENT Build Version String.

2.24.1 Detailed Description

Defines API that obtains the values of the various version identifiers.

Copyright

Description:

Provide version identifiers for the cFE Platform Support Packages (PSP). See cfsversions for version and build number and description∗∗ GSC-18128-1, "Core Flight Executive Version 6.7"

Limitations, Assumptions, External Events, and Notes:

None

2.24.2 Macro Definition Documentation

2.24.2.1 #define CFE_PSP_IMPL_VERSION CFE_PSP_IMPL_BUILD_BASELINE "+dev" CFE_PSP_IMPL_STR(CFE_PSP_IMPL_BUILD_NUMBER)

DEVELOPMENT Build Version Number.

Baseline git tag + Number of commits since baseline.

See cfsversions for format differences between development and release versions.

2.24.2.2 #define CFE_PSP_IMPL_VERSION_STRING

Value:

```
" PSP Development Build " CFE_PSP_IMPL_VERSION /* Codename for current development */ \
    ", Last Official Release: psp v1.4.0"          /* For full support please use this version */
```

DEVELOPMENT Build Version String.

Reports the current development build's baseline, number, and name. Also includes a note about the latest official version.

See cfsversions for format differences between development and release versions.

Index