Global Varible

Declared at `xmconf.h`. Their data is assigned by xmcparser and xml tools. They are most likely to be read-only in XM kernel.

xmcPartitionTab

Declaration

```
//file core/kernel/setup.c
struct xmcPartition *xmcPartitionTab;
```

As described in the next section. `xmcPartition` is loaded according to XM partitions' configurations and attributes. It is indexed by numerical id.

```
    struct xmcPartition {
        xmId_t id;
        xm_u32_t nameOffset;
        xm_u32_t flags;
    #define XM_PART_SYSTEM 0x100
    #define XM_PART_FP 0x200
        xm_u32_t noVCpus;
        xm_u32_t hwIrqs;
        xm_s32_t noPhysicalMemoryAreas;
        xm_u32_t physicalMemoryAreasOffset;
        xmDev_t consoleDev;
        struct xmcPartitionArch arch; //empty
        xm_u32_t commPortsOffset;
        xm_s32_t noPorts;
        struct xmcHmSlot hmTab[XM_HM_MAX_EVENTS];
        xm_u32_t ioPortsOffset;
        xm_s32_t noIoPorts;
        struct xmcTrace trace;
        struct xmcPartIpvi {
            xm_u32_t dstOffset;
            xm_s32_t noDsts;
        } ipviTab[CONFIG_XM_MAX_IPVI];
    };
```

Description

An array of `xmcPartiton`. Length of this array is `xmcTab.noPartitions`. `xmcPartition` struct consists of the id of partition, number of virtual CPUs assigned to this partition, communication ports of the partition, consoleDev, etc.. It is the detailed representation of XtratuM partition transalted from XML files.

One of the attribute `xmcPartitionArch` is empty.

Initialization

Initialization is done using xmcparser and xml tools.

Functions

1. SetupPartitions

   xmcPartitionTab[e].noPhysicalMemoryAreas is used for print information about physical memory area in this function.

2. CreatePartition

   Assign xmcPartition to a partition_t p, a certain element in partitionTab array.

   For each virtual CPU, allocate one thread to partition, with flags cleared and timers allocated.

xmcMemRegTab

Declaration

```
//file core/kernel/setup.c
struct xmcMemoryRegion *xmcMemRegTab;
```

```
    struct xmcMemoryRegion {
        xmAddress_t startAddr;
        xmSize_t size;
    #define XMC_REG_FLAG_PGTAB (1<<0)
    #define XMC_REG_FLAG_ROM (1<<1)
        xm_u32_t flags;
    };
```

Description

An array of `xmcMemoryRegion`. The length of this array is `xmcTab.noRegions`, which is the number of regions of XtratuM. `xmcMemoryRegion` struct consists of the start address, size of the memory region and corresponding flags of it.

Initialization

Initialized by parser and xml tools.

Functions

1. PmmFindAnonymousPage

   Binary search for a certain address.

2. PmmFindPage & PmmFindArea

   Similar as above

3. PmmResetPartition

   Called at function `ResetPartition`. This function reset partition's physical memory's type and counter to PPAG_STD and 0.

```
    page=&physPageTab[memArea->memoryRegionOffset][(addr-memRegion->startAddr)>>PAGE_SHIFT];
```

```
is used to find addr located page.
```

1. SetupPhysMM

   Create empty physical memory of size:
   `//e for one of current xmcTab.noRegions GET_MEMZ(physPageTab[e], sizeof(struct physPage)*(xmcMemRegTab[e].size/PAGE_SIZE))`
   and init spinlock for the area

---

xmcPhysMemAreaTab

Declaration

```
//file core/kernel/setup.c
struct xmcMemoryArea *xmcPhysMemAreaTab;
```

```
    struct xmcMemoryArea {
        xm_u32_t nameOffset;
        xmAddress_t startAddr;
        xmAddress_t mappedAt;
        xmSize_t size;
    #define XM_MEM_AREA_SHARED (1<<0)
    #define XM_MEM_AREA_UNMAPPED (1<<1)
    #define XM_MEM_AREA_READONLY (1<<2)
    #define XM_MEM_AREA_UNCACHEABLE (1<<3)
    #define XM_MEM_AREA_ROM (1<<4)
    #define XM_MEM_AREA_FLAG0 (1<<5)
    #define XM_MEM_AREA_FLAG1 (1<<6)
    #define XM_MEM_AREA_FLAG2 (1<<7)
    #define XM_MEM_AREA_FLAG3 (1<<8)
    #define XM_MEM_AREA_TAGGED (1<<9)
    #define XM_MEM_AREA_IOMMU (1<<10)
        xm_u32_t flags;
        xm_u32_t memoryRegionOffset;
    };
```

Description

An array of `xmcMemoryRegion`. The size of the array is the summ of `xmcPartitonTab[0~xmcTab.noPartitions-1].noPhysicalMemoryAreas`. SDEV_SEEK_ENDtruct `xmcMemoryArea` consists of its starting address, mapped address, flags, as well as the size of this memory area.

This array is used mainly to record the memory size allocation. Array `memBlockData` is used to keep tracking the usage of memory of a `kDevice_t`.

Initialization

Initialized by parser and xml tools.

Functions

1. ReadMemBlock & WriteMemBlock

   //file core/drivers/memblock.c // In ReadMemBlock, there may be error if no mmu is defined on SPARC arch These two functions are more or less the same. Frist check if there is enough space left, then do memory copy and update memory usage correspondingly.

2. SeekMemBlock

   Pass seek operation, DEV_SEEK_START, DEV_SEEK_CURRENT and DEV_SEEK_END.

3. InitMemBlock

   VmMapPage virtual memory paging operation here

4. SetupVmMap

   //file core/kernel/arch/vmmap.c

   Set flags and initial value of `_ptdL3`

5. PmmFindPage & FindAddr & FindArea & ResetPartition

   //file core/kernel/mmu/physmm.c

   Uses xmcPhysMemAreaTab's memory region information.

6. SetupPageTable

   //file core/kernel/mmu/vmmap.c

   same as above

7. SetupPartitions

   //file setup.c

   same as above

---

xmcCommChannelTab

Declaration

```
//file core/kernel/setup.c
struct xmcCommChannel *xmcCommChannelTab;
```

```
    struct xmcCommChannel {
    #define XM_SAMPLING_CHANNEL 0
    #define XM_QUEUING_CHANNEL 1
    #if defined(CONFIG_DEV_TTNOC)||defined(CONFIG_DEV_TTNOC_MODULE)
    #define XM_TTNOC_CHANNEL 2
    #endif
        xm_s32_t type;
        union {
            struct {
                xm_s32_t maxLength;
                xm_s32_t maxNoMsgs;
            } q;
            struct {
                xm_s32_t maxLength;
                xm_u32_t validPeriod;
                xm_s32_t noReceivers;
            } s;
    #if defined(CONFIG_DEV_TTNOC)||defined(CONFIG_DEV_TTNOC_MODULE)
            struct {
                xm_s32_t maxLength;
                xm_u32_t validPeriod;
                xm_s32_t noReceivers;
                xmId_t nodeId;
            } t;
    #endif
        };
    };
```

Description

An array of `xmcCommChannel` with size of `xmcTab.noCommChannels`. Struct `xmcCommChannel` contains type and union of xm channel.

Initialization

Initialized by parser and xml tools.

Functions

It is used in file core/objects/commports.c mainly (not considering ttnocports.c here). This struct only provide configuration details.

---

xmcCommPorts

Declaration

```
//file core/kernel/setup.c
struct xmcCommPort *xmcCommPorts;
```

```
    struct xmcCommPort {
        xm_u32_t nameOffset;
        xm_s32_t channelId;
    #define XM_NULL_CHANNEL -1
        xm_s32_t direction;
    #define XM_SOURCE_PORT 0x2
    #define XM_DESTINATION_PORT 0x1
        xm_s32_t type;
    #define XM_SAMPLING_PORT 0
    #define XM_QUEUING_PORT 1
    #define XM_TTNOC_PORT 2
    #if defined(CONFIG_DEV_TTNOC)||defined(CONFIG_DEV_TTNOC_MODULE)
        xmDev_t devId;
    #endif
    };
```

Description

Similar as above.

Initialization

Functions

---

xmcIoPortTab

Declaration

```
//file core/kernel/setup.c
struct xmcIoPort *xmcIoPortTab;
```

```
    struct xmcIoPort {
        xm_u32_t type;
    #define XM_IOPORT_RANGE 0
    #define XM_RESTRICTED_IOPORT 1
        union {
            struct xmcIoPortRange {
                xmIoAddress_t base;
                xm_s32_t noPorts;
            } range;
            struct xmcRestrictdIoPort {
                xmIoAddress_t address;
                xm_u32_t mask;
    #define XM_DEFAULT_RESTRICTED_IOPORT_MASK (~0)
            } restricted;
        };
    };
```

Description

A single port with restrictions on the bits the the partition is allowed to write in. Only those bits that are set in the mask, can be modified by the partition.

Initialization

Functions

1. IoPortLogSearch

   Iterate among the partition’s IO ports and return found Port’s mask.

2. SparcIoOutportSys

   Use the IoPortLogSearch shown above.

---

xmcRsvMemTab

Declaration

```
//file core/kernel/setup.c
struct xmcRsvMem *xmcRsvMemTab;
```

```
    struct xmcRsvMem {
        void *obj;
        xm_u32_t usedAlign;
    #define RSV_MEM_USED 0x80000000
        xm_u32_t size;
    } __PACKED;
```

Description

An array that keeps recording which memory region is reserved/used.

Initialization

Initialized by parser and xml tools.

Functions

1. InitRsvMem

   //file core/kernel/rsvmem.c

```
    void InitRsvMem(void) {
    //mark as unused
        xm_s32_t e;
        for (e=0; xmcRsvMemTab[e].obj; e++)
            xmcRsvMemTab[e].usedAlign&=~RSV_MEM_USED;
    }
```

1. AllocRsvMem

   Use for-loop to iterat among memory objects one by one. If currrent memory's size is equal to required memory size, then mark the memory as used and return its address.

   This function is used by `GET_MEMA` and `GET_MEMAZ` functions, which are used for allocating thread, stack and page memory.

---

xmcBootPartTab

Declaration

```
//file core/kernel/setup.c
struct xmcBootPart *xmcBootPartTab;
```

```
    struct xmcBootPart {
    #define XM_PART_BOOT 0x1
        xm_u32_t flags;
        xmAddress_t hdrPhysAddr;
        xmAddress_t entryPoint;
        xmAddress_t imgStart;
        xmSize_t imgSize;
        xm_u32_t noCustomFiles;
        struct xefCustomFile customFileTab[CONFIG_MAX_NO_CUSTOMFILES];
    };
```

Description

This array stores information about partition boot image address, entry point, details about the custom files. //TODO

Initialization

Initialized by parser and xml tools.

Functions

1. SetupLdr

   `xmcBootPartTab` provides image start address. The partition image mapping is setted in this function.

2. ResetPartition

   Reset page table using image handler with address `hdrPhysAddr`. Reset partition's thread (Warm and boot situation) with entryPoint in `xmcBootPartTab`.

---

xmcRswInfo

Declaration

```
//file core/kernel/setup.c
struct xmcRswInfo *xmcRswInfo;
```

Description

Not in use

Initialization

Functions

---

xmcDstIpvi

Declaration

```
//file core/kernel/setup.c
struct xmcRswInfo *xmcRswInfo;
```

```
    struct xmcRswInfo {
        xmAddress_t entryPoint;
    };
```

Description

Ipvi stands for *Inter-Partition Virtual Interrupts*. The `XM_raise_ipvi()` hypercall generates an virtual interrupt to one or several partitions as speficied in the configuration file.

Initialization

Initialized by parser and xml tools.

Functions

1. hypercall RaiseIpviSys

   This function is the implementation of `XM_raise_ipvi()` hypercall. This hypercall generates an virtual interrupt to one or several partitions as speficied in the configuration file. The link between the partition that generates the interrupt and the receiver partitions is specified in the channel section of the configuration file.

2. hypercall RaisePartitionIpviSys

   This function has similar implementation as above. However, the return values of them are different. Set irq pending if and only if `xmcDisIpvi[ipvi->disOffset + e] == partitionId`.

---

xmcStringTab

Declaration

```
//file core/kernel/setup.c
xm_s8_t *xmcStringTab;
```

Description

This array of string keeps the name of partitions and plans.

Initialization

Initialized by parser and xml tools.

Functions

1. hypercall GetGidByNameSys

   This function is the implementation of XM_get_gid_by_name. Returns in the identifier of an entity as defined in the configuration file by providing the entity name string.

2. SetupPct

   //file core/kernel/kthread.c

   Setup partiton ctrl requires the name of partition.

   Setup memory area requires the name of `xmcMemArea->nameoffset`.

3. file core/objects/commports.c

   Finding port is using the name of ports.

---

xmcVCpuTab

Declaration

```
//file core/kernel/setup.c
struct xmcVCpu *xmcVCpuTab;
```

```
    struct xmcVCpu{
        xmId_t cpu;
    };
```

Description

This array is used to store the IDs of CPUs for each partition.

Initialization

Initialized by parser and xml tools.

Functions

1. hypercall ResumeVCpuSys

   Use more informaiton about current partition’s smp vcpu. Send ipi to the cpu that is not running current thread.

2. hypercall RaisePartitionIpviSys

   Similar as above. SMP support

3. hypercall RaiseIpviSys

   Similar as above. SMP support

4. CreatePartition

5. SetupPct

   For index VCpu scheduling policy

6. ResetKThread

   Reset current thread only. If smp, then can keep scheduling on other cores.