Implementing Segmented Virtual Memory Support in a System Model Using MMU (TCC-MEM-MMU)

Introduction

The TCC-MEM-MMU component allows basic segmented memory to be modeled in an SSDL system. This technical note describes how to include MMU in your system.

Including MMU in your system

MMU component can be included in your system by using the following code:

```
;device models
model device MMU tc_core_TCCMMU.so
; Define MMU of 5MB with each segment is of size 0x20000000, Leaving 3 Bits(3bits
-> 8 segments) in 32bit
; address (nabits=32) for Segment Index (lines != 0) Physical Segment Address
(lines = 8).
component MMU segmentation_unit nabits=32 size=0x20000000 mode=rw segments=8
lines=0 words=0x20000000

;; Segmented Memory (VM) Appears at Address 0x0 of Processors Unified Memory Space
map memory core:address.0x00000 segmentation_unit:STDRAM rw

; Physical Memory Appears at Address 0x10000000 of MMU's Address Space
map memory segmentation_unit:u.0x10000000  mem:STDRAM rw

; Map Various MMU Registers to Processor (Physical) Address Space

map memory core:address.0xfffc0000 segmentation_unit:VSR rw
map memory core:address.0xfffc0004 segmentation_unit:PSR rw
map memory core:address.0xfffc0008 segmentation_unit:SMR rw
map memory core:address.0xfffc000c segmentation_unit:SSR rw
map memory core:address.0xfffc0010 segmentation_unit:STEN rw
map memory core:address.0xfffc0014 segmentation_unit:DEBUGR rw
map memory core:address.0xfffc0018 segmentation_unit:ATUMR rw
map memory core:address.0xfffc001c segmentation_unit:MMUER rw
; Map interrupt register to MMUER
map register segmentation_unit:MMUER core:r33
```

The relevant parameters are described in the table.

Parameter	Description
nabits	This is the number of bits addresses by the north side bridge of the CPU.
size	This is the size of MMU that is required
mode	This is the mode used for the MMU. It can take the values of ro, rw, wo.
segments	This specifies the number of segments required for the MMU. The segment being accessed may be set using the STEN register in the player command file. The segment number goes from 0 to segment – 1.
lines	This is specified when the paging is used in MMU. This specifies the number of lines required.
words	This specifies the size of each segment in multiples of words.

Configuring the MMU

Different segments can be set by appropriately setting MMU registers as indicated below. Refer to Appendix A for detailed description of the MMU registers.

```
; set ATUMR so that MMU is enabled
memset 0xfffc0018 0x01000000
; set DEBUGR so that memory translation trace is enabled
memset 0xfffc0014 0x1
; Segment 1: 0 -> 0x10000000
; vsr (Virtual Address)
memset 0xfffc0000 0
; psr (Physical Address)
memset 0xfffc0004 0x00000010
; smr (Segment Mode Read/Write)
memset 0xfffc0008 0x11000000
; ssr (Segment Size)
;memset 0xfffc000c 0x00000001
memset 0xfffc000c 0x01000000
; sten
memset 0xfffc0010 0
memset 0x2 1
```

Note
When setting the MMU registers, ensure that the value specified is byte swapped (DR 41555).

Switching Segments Dynamically

The following sample code demonstrates how the segments can be switched dynamically. This can be used in a kernel code to switch between processes which are mapped to different segments.

```
/* A minimal kernel that switches between two processes by changing the segment
mapping */

#include "PICO_Kernel.h"
#include "Timer.h"

timer_t timer = (timer_t)0xfffa0000;

// MMU is mapped to 0xfffc0000
mmu_device_t mmu = (mmu_device_t)0xfffc0000;

/* Process 1: vsr = 0x200000, psr  = 0x200000, rw mode, size=0x010000,
   segment num=1
*/
struct mmu_config_t p1_config = { 0x200000, 0x200000, 0x11, 0x01, 0x01};

/* Process 2: vsr = 0x200000, psr  = 0x400000, rw mode, size=0x010000,
   segment num=1 */
struct mmu_config_t p2_config = { 0x200000, 0x400000, 0x11, 0x01, 0x01};

process_t proc_list[MAX_PROCESS] = {
       { (void *)0x200000, &p1_config },
       { (void *)0x200000, &p2_config }
};

int current_process = 0;

void dummy()
{
}

void switch_context()
{
        current_process++;

      if (current_process >= MAX_PROCESS)
          current_process = 0;

        /* Set MMU context to current process */
        mmu_set_context(mmu, proc_list[current_process].segment_map);

      /* Call entry point of current process */
      (*(proc_list[current_process].entry))();
}

int kernel_main()
{
      /* Initialize the MMU */
      mmu_init(mmu);

      Timer_enable(timer);

        /* Set MMU context to current process */
        mmu_set_context(mmu, proc_list[current_process].segment_map);
```

```
        /* Call entry point of current process */
        (*(proc_list[current_process].entry))();

        while (1);

        return 0;
}
```

In the above code, mmu_init is a function defined in the MMU driver for initializing the MMU and mmu_set_context sets the MMU segment registers to the specified values. The kernel_main() function initializes the MMU and sets context to p1_config. Then it calls process1 entry point using the pointer to process 1 entry. Next, the kernel goes into an infinite loop. The kernel interrupt handler for timer interrupt irq_handler(), checks for a specified number of interrupts before switching context to the next process in the process list. The MMU context of the new process is loaded into the MMU and its entry point is called.

Sample Code for Process 1

```
#include "tc_core_Console.h"

console_device_t console = (console_device_t)0xffffd000;

int main()
{
        console_write(console, "Process 1");

    return 0;
}
```

Sample Code for Process 2

```
#include "tc_core_Console.h"

console_device_t console = (console_device_t)0xffffd000;

int main()
{
        console_write(console, "Process 2");

    return 0;
}
```

Player Command File

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; SSDL Description for system: PICO_System
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; include board description
inc ../PICO_Board/board.ssdl

; set ATUMR so that MMU is enabled
memset 0xfffc0018 0x1

; To enable memory translation trace set the below address (DEBUGR) to 1
memset 0xfffc0014 0x0

; Segment 0: Kernel: 0 -> 0x0
; vsr (Virtual Address)
memset 0xfffc0000 0
; psr (Physical Address)
memset 0xfffc0004 0x0
; smr (Segment Mode Read/Write)
memset 0xfffc0008 0x00000011
; ssr (Segment Size)
memset 0xfffc000c 0x01
; sten
memset 0xfffc0010 0

;; include software description
load ../PICO_Kernel/PICO_Kernel.x

; Break at dummy()
b 0x8000

; Segment 1: Process1: 0x00200000 -> 0x200000
; vsr (Virtual Address)
memset 0xfffc0000 0x200000
; psr (Physical Address)
memset 0xfffc0004 0x200000
; smr (Segment Mode Read/Write)
memset 0xfffc0008 0x11
; ssr (Segment Size)
memset 0xfffc000c 0x01
; sten
memset 0xfffc0010 0x01

; Load process 1
load ../PICO_App1/PICO_App1.x

; Segment 2: Process2: 0x00200000 -> 0x400000
; vsr (Virtual Address)
memset 0xfffc0000 0x200000
; psr (Physical Address)
memset 0xfffc0004 0x400000
```

```
; smr (Segment Mode Read/Write)
memset 0xfffc0008 0x11
; ssr (Segment Size)
memset 0xfffc000c 0x01
; sten
memset 0xfffc0010 0x01

; Load process 2
load ../PICO_App2/PICO_App2.x

; Set pc to kernel_main
set pc 0x80a0

; Set stack pointer
set r13 0x10000
set r13_irq 0x20000


srun

quit
```

Porting MMU sample to other targets

The MMU sample can be ported to other targets. This requires the following changes:

1. The SSDL description (PICO_Board/board.ssdl) should be suitably modified to reflect the target system architecture and MMU configuration
2. The Player command file (PICO_System/system.ssdl) should be updated to setup the required segments
3. The kernel code (PICO_Kernel/PICO_Kernel.c) should be modified to reflect the segment mapping specified in system.ssdl
4. The interrupt vector table should be setup as required (PICO_Kernel/init.s)

Appendix A – MMU Register Description

The table below provides information on MMU register description.

Register Name	Description	Example
VSR	Virtual Segment Register. Setting this enables the start address of virtual address that is being mapped to the physical address immediately given after memset for VSR in the player command file.	memset <VSR address> <address to be mapped> ;memset 0xfffc0000 0x20000000 memset 0xfffc0000 0x00000020
PSR	Physical Segment Register. This is required for mapping virtual address to the physical address. This value is added to the physical address base value given for the particular segment. This is set in the player command file as shown.	memset <PSR address> <address to be mapped> ;memset 0xfffc0004 0x10000000 memset 0xfffc0004 0x00000010
SMR	Segment Mode Register. This is used to set the mode of operation whether read or write or rw. This is set in the player command file as shown.	memset <SMR address> <ACCESS> memset 0xfffc0008 0x11 0x1 -> read only 0x01-> write only 0x11-> read/write
SSR	Segment Size Register. This is used to set the size of each segment in multiples of words. This may be set in the player command file as shown.	memset <SSR address> <Size in words> memset 0xfffc000c 0x01000000
STEN	Segment table entry number. This is used to specify which segment is being accessed. Setting this would write above mentioned registers to the segment table. All these five registers are to be given in the same order as VSR, PSR, SMR, SSR, STEN. This may be set in the player command file as shown.	memset <STEN address> <Segment number> memset 0xfffc0010 0x1
ATUMR	This is used to enable and disable MMU. This may be set in the player	memset <ATUMR address> <activation> ;memset 0xfffc0018 1

	command file as shown.	`memset 0xfffc0018 0x01000000`
		
		`0 -> Disable`
		`1 -> Enable`

DEBUGR	This is used for debugging. Enabling this would trace the address translation and displays it in the simulator. This may be set in the player command file as shown.	`memset <DEBUGR address> <activation>`
		`memset 0xfffc0014 1`
		
		`0 -> Disable`
		`1 -> Enable`

MMUER	This register indicates the exception status. Value of 0x1 indicates protection violation and 0x4 indicates invalid address.	

Reference

For more information, refer to SANKHYA Teraptor User Guide and Reference Manual.

Technical Support

Contact support@sankhya.com for more information.

SANKHYA™

Sankhya Technologies Private Limited
Contact:
+1 408-556-9757 (USA)
+91 94449 72818 (India and South Asia)
http://sankhya.com/contact.html