Scenario Description – SMMU Invalidation

Date of Issue: 28/11/2014

Confidentiality: Confidential

Abstract

This document contains a test scenario description supported by a block diagram that shows key blocks of the system it was run on and snippets of code as exhibits. This is meant to serve as an example for the reader to be able to create something similar in their environment, and not mean to run as is.

Contents

1 ABOUT THIS DOCUMENT

1.1 Purpose

The purpose of this document is to provide a reference to the reader, for a system setup and supporting code snippets to exercise DVM transaction with the SMMU 500.

It is important to note that the code snippets (in Section 5) are only in the form of exhibits to support the Test Scenario and Test Pseudo Code described in Section 3 and Section 4 respectively.

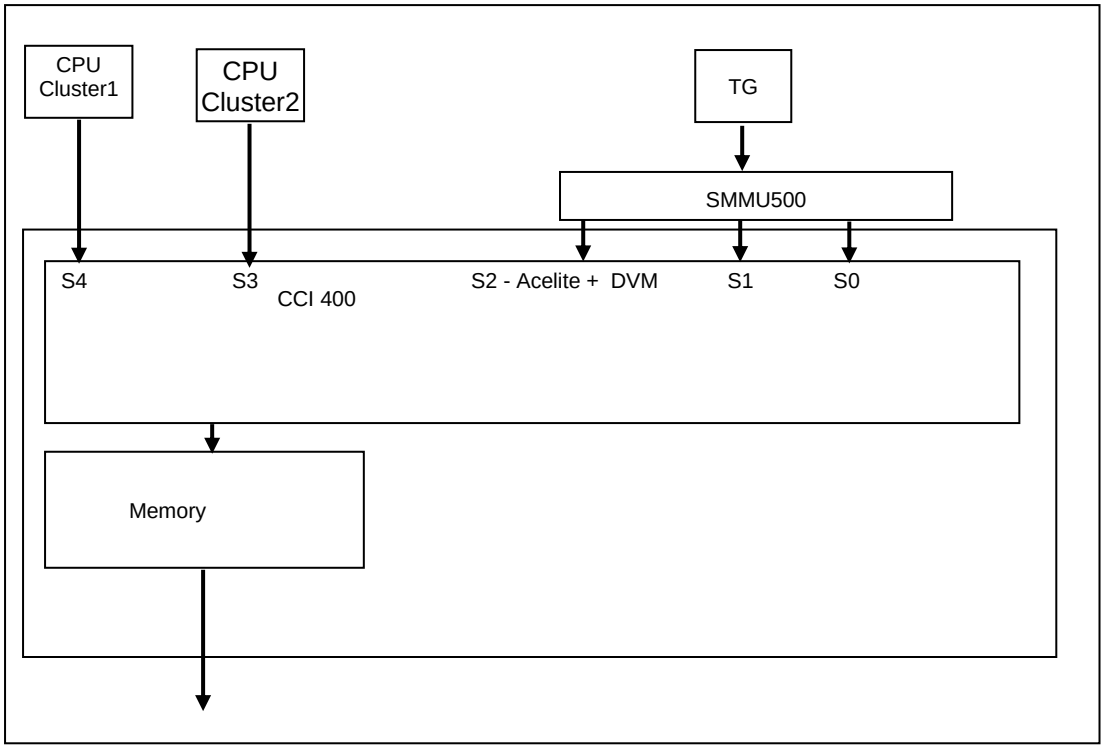
The code snippets, are only intended to be read, and are not meant to be used as-is.

This document has no knowledge or makes no assumptions about the end users system, software or test environments.

2 SYSTEM CONFIGURATION

We assume a system with the following configuration.

a) V8 CPU's in the CPU Clusters
b) TCU interface is connected to an Acelite + DVM interface on interconnect.
c) The SMMU used is a 4x4 config



3 TEST SCENARIO

1. The test scenario uses 2 Virtual Address (VA) regions, namely VA-A, VA-B and 3 physical Address regions, namely A, B and C.
2. Initially, the following Virtual Address (VA) to Physical address mapping is created in the page tables:
    a. VA-A -> A
    b. VA-B -> B
3. The traffic generator (TG) performs a Memcopy between VA-A and VA-B.
4. The virtually address region VA-B is remapped to Physical address region C in the page tables. This step involves invalidation of the earlier mapping between VA-B and B.
5. The traffic generator (TG) performs a Memcopy between VA-A and VA-B again.
6. Now the contents of A and C should be same, if the remapping and corresponding invalidation happened correctly.
7. A mismatch in A and C contents is flagged as failure.

4 TEST PSEUDO CODE

1) Setup page table entries for regions Physical Addresses A and B. (VA-A - > A, VA-B -> B)
2) Initialize region A with a specific pattern.
3) Program SMMU (as shown in Exhibit -1)
   a) Read SMMU IDR0 to get the number of SMR's
   b) Read SMMU IDR1 to get the number of context banks.
   c) Calculate Number of pages From IDR1, 2(SMMU_IDR.NUMPAGENDXB + 1)
   d) Calculate the Context bank Base: SMMU_BASE + NUM_PAGES * PAGE_SZ
   e) Initialize all SMR's and CB's to zero.
   f) Program CR0 register.
4) Program the SMR register and S2CR register to map the stream and set in translation mode. (Exhibit - 2)
5) Program TTBCR/TCR and TTBR0 reg.
6) Program CCI DDCR to allow DVM's to the interface where TCU is connected.
7) Start the traffic generator. Do Memcopy between A and B. (Virtual Address, VA-A to VA-B)
8) Compare contents of A and B, to check that they are identical
9) Compare contents of A and C, to check that they are not identical
10) Change the translation such that, Virtual Address, VA-B is pointing to C.
11) Do tlbi to invalidate the entry in cache. (Exhibit – 3)
12) Restart the transfer from VA-A to VA-B.
13) Compare contents of A and C and flag the results. (Pass if contents are identical; else Fail)

5 EXAMPLE ROUTINES

5.1 Sample code snippet for SMMU initialization - (Exhibit -1)

```
int32_t    smmu_dev_init(regaddr_t base, struct smmu_cfg_params *params, struct smmu_dev_info *priv)
{
    uint32_t regval, i, pageidx;

  /*
   * Do not intialize if device is enabled. It may be in use
   */

  regval = SMMU_REG_READ32(base + SMMU_CR0_OFF);
  if((regval & SMMU_CR0_CLIENTPD) == 0)
       return -SMMU_EBUSY;

 priv->smmu_base = base;

 /*  Read the smmu version */
 priv->smmu_ver = SMMU_REG_READ32(priv->smmu_base + SMMU_IDR7_OFF) &
SMMU_IDR7_REV_MASK;
```

```
/* Get the number of smrs */
regval = SMMU_REG_READ32(priv->smmu_base + SMMU_IDR0_OFF);
priv->num_smr = regval & SMMU_IDR0_NUMSMRG_MASK;

regval = SMMU_REG_READ32(priv->smmu_base + SMMU_IDR2_OFF);
priv->ias = regval & SMMU_IDR2_IAS_MASK;
priv->addr_sz_mask = (0x100000000L << (priv->ias << 2)) - 1;

/* Get the page size */
 regval = SMMU_REG_READ32(priv->smmu_base + SMMU_IDR1_OFF);
 if(regval & SMMU_IDR1_PAGESIZE)
   priv->page_sz = 0x10000;    /* 64 KB */
 else
   priv->page_sz = 0x1000;     /* 4 KB */

 /* Get the number of Context banks */
 priv->num_cb = regval & SMMU_IDR1_NUMCB_MASK;


 /*
  * Calculate cb_base
  */
 pageidx = (regval & SMMU_IDR1_NUMPAGENDXB_MASK) >> SMMU_IDR1_NUMPAGENDXB_POS;
 priv->num_pages = 1 << (pageidx + 1);
 priv->cb_base = priv->smmu_base + priv->num_pages * priv->page_sz;

 /*
  * Ensure global interrupts are disabled
  */
 smmu_dev_dis_intr(priv);

 /* Ensure all context interrupts are disabled */
 for( i = 0 ; i < priv->num_cb ; i++)
 {
   smmu_cb_dis_intr(priv, i);
   priv->free_cb_list[i] = i;  /* Initialize allocator indexes */
 }
 priv->free_cb_idx = 0;

 /*
  * Clear all global faults
  */
 SMMU_REG_WRITE32((priv->smmu_base + SMMU_GFSR_OFF), 0xFFFFFFFF);

 /*
  * Clear all SMR
  */
```

```
for(i = 0; i < priv->num_smr; i++)
{
  regval = SMMU_REG_READ32(priv->smmu_base + SMMU_SMR_BASE_OFF + 4*i);
  regval &= ~SMMU_SMR_VALID;
  SMMU_REG_WRITE32(priv->smmu_base + SMMU_SMR_BASE_OFF + 4*i, regval);
  priv->free_smr_list[i] = i;  /* Initialize allocator indexes */
}
priv->free_smr_idx = 0;

/*
 * Clear all CB
 */
for(i = 0; i < priv->num_cb; i++)
{
  regval = SMMU_REG_READ32(priv->smmu_base + SMMU_S2CR_BASE_OFF + 4*i);
  regval &= ~SMMU_S2CR_CBAR_TYPE_MASK;
  regval |= SMMU_S2CR_CBAR_TYPE_FAULT;
  SMMU_REG_WRITE32((priv->smmu_base + SMMU_S2CR_BASE_OFF + 4*i), regval);
}

/*
 * Invalidate All TLB
 */
smmu_inv_gtlb(priv, 0, 0);

/*
 * Check for stall support
 */
priv->is_stall_supp = 0;
regval = SMMU_REG_READ32(priv->smmu_base + SMMU_CR0_OFF);
regval |= SMMU_CR0_GSE;
SMMU_REG_WRITE32(priv->smmu_base + SMMU_CR0_OFF, regval);
regval = SMMU_REG_READ32(priv->smmu_base + SMMU_CR0_OFF);
priv->is_stall_supp = !!(regval & SMMU_CR0_GSE);

/*
 * Config SMMU_CR0
 */
regval = SMMU_REG_READ32(priv->smmu_base + SMMU_CR0_OFF);

  regval = ((params->glb_flt_rep_en << SMMU_CR0_GFRE_POS)
    | (params->glb_cfg_flt_rep_en << SMMU_CR0_GCFGFRE_POS)
    | (params->stall_dis << SMMU_CR0_STALLD_POS)
    | (params->glb_stall_en << SMMU_CR0_GSE_POS)
    | (params->stm_map_tbl_no_match_cfg << SMMU_CR0_SMTNMC_POS)
    | (params->stm_cft_flt_cfg << SMMU_CR0_SMCFCFG_POS)
    | (params->vmid_priv_namespace_en << SMMU_CR0_VMIDPNE_POS)
    | (params->priv_tlb_maintenance << SMMU_CR0_PTM_POS)
```

```
    | (params->mem_attr << SMMU_CR0_MEMATTR_POS)
    | (params->mt_cfg << SMMU_CR0_MTCFG_POS)
    | (params->sh_cfg << SMMU_CR0_SHCFG_POS)
    | (params->rd_alloc_cfg << SMMU_CR0_RACFG_POS)
    | (params->wr_alloc_cfg << SMMU_CR0_WACFG_POS));
  SMMU_REG_WRITE32(priv->smmu_base + SMMU_CR0_OFF, regval);

  /*
   * Enable SMMU
   */
  regval = SMMU_REG_READ32(priv->smmu_base + SMMU_CR0_OFF);
  regval &= ~SMMU_CR0_CLIENTPD;
  SMMU_REG_WRITE32(priv->smmu_base + SMMU_CR0_OFF, regval);

  return SMMU_SUCCESS;
}
```

5.2 Sample code snippet for mapping stream - (Exhibit - 2)

```
int32_t   smmu_map_stream(struct smmu_dev_info *priv, struct smmu_s2cr_params *params, uint32_t
strm_flags)
{

  regaddr_t addr=0x0;
  uint32_t regval = 0x0;


  /* get a  Free smr */
  uint32_t   smrnum = smmu_get_smr_idx(priv);



  if(smrnum == SMMU_INVALID_SMR)
    return -SMMU_ENOMEM;

  /*
   *   Program corresponding S2CR reg
   */

  addr = priv->smmu_base + SMMU_S2CR_BASE_OFF + 4 * smrnum;

  regval = (params->cb_idx & SMMU_S2CR_CBAR_CBNDX_MASK)
   | ((params->mem_attr  << SMMU_S2CR_CBAR_MEM_ATTR_POS) &
SMMU_S2CR_CBAR_MEM_ATTR_MASK)
   | ((params->mem_type_cfg << SMMU_S2CR_CBAR_MTCFG_POS) & SMMU_S2CR_CBAR_MTCFG_MASK)
   | ((params->shared_cfg << SMMU_S2CR_CBAR_SHCFG_POS) & SMMU_S2CR_CBAR_SHCFG_MASK);
  SMMU_REG_WRITE32(addr, regval);
```

```
  /* Program SMR register */
  addr = priv->smmu_base + SMMU_SMR_BASE_OFF + 4 * smrnum;
  regval = strm_flags;
  SMMU_REG_WRITE32(addr, regval);

  smmu_map_en(priv, smrnum);

  return smrnum;
}
```

5.3 Sample routine for V8 invalidation by VA - (Exhibit - 3)

```
  __asm void cpu_invlvae3is (uint64_t x2) {
              SUB  sp, sp, #0x10
              STP  x29, x30, [sp,#0]
              MOV  x29, sp
              MRS  x2, SCR_EL3
              MOV  x1, #0x01
              ORR  x2, x2, x1
              MSR  SCR_EL3, x2
              ISB
              TLBI IPAS2E1IS, x0
              DSB  SY
              ISB
              MOV  x1, #0x1
              BIC  x2, x2, x1
              MSR  SCR_EL3, x2
              ISB
              LDP  x29,x30,[sp,#0]
              ADD  sp,sp,#0x10
              RET

            } ; //TLBIVAE3IS
```