Embest Technology

# AMP Reference Design User Manual

(base on Cyclone V EVM Board REV C)

## Revision History:

| Version | Date | Description |
|---|---|---|
| 1.0 | 2014-01-03 | Original Version |
| 1.1 | 2014-01-15 | Add AXI single-access-time test, amend some descriptions |
| 1.2 | 2014-02-20 | Add ACP program guide. Add sys system user space sgi interrupt trigger. |
| 1.3 | 2014-04-29 | Add AMP app development tips |

## Abbrevations:

| Abbrevations | Full Name |
|---|---|
| **AMP** | **A**symmetric **M**ulti**p**rocessing |
| **BM** | **B**are **M**etal |
| **IL** | **I**nterrupt **L**atency |
| **SGI** | **S**oftware-**G**enerated **I**nterrupt |
| **IPI** | **I**nter-**P**rocessor **I**nterrupt |

## Software List:

- **u-boot-altera-2012.10.tar.bz2:**u-boot
- **linux-altera-3.7-a.tar.bz2:**linux kernel
- **bm_reference_design.zip:**bm project
- **amp_app_mmap.zip:**amp app mmap project
- **bm.rbf:** FPGA config file
- **fpga_project.zip:** FPGA QSYS design files

# Table of Contents

# 1 Features of AMP Reference Design

- Be able to boot from QSPI
- Be able to boot from SD card
- SupportBMbooting first
- Less than 50ms from 1st instruction of preloader to BM main entry
- BM is responsible for loading FPGA image
- BM can enable MMU, L1 instruction cache, L1 data cache, and L2 cache to boost system performance
- Cpu0 and cpu1 both operateat 800MHz with 2.5 DMIPS/MHz per core
- CPU0 and CPU1 can send/receive IPI(Inner Processor Interrupt/Communication) to/from each other(phase 2)
- CPU0 and CPU1 can communicate with each other by polling
- SCU enabled for MPCORE cache coherence
- Support data transfer by ACP
- Supportdata transfer by DMA
- Support L2 cache Lock by Master and the size can be changed by configuring parameter
- Peripheral device can be set free and assigned to cpu0 or cpu1(through GIC interrupt distributor)
- Support Ethernet on the HPS(Linux)
- FPGA works as slave at AXI bridge
- Suport data transfer from FPGA to HPS by ACP
- Memory-to-memory DMA transfer from ARM to FPGA and verify the data transferred in ARM(the size of data block can be changedusing MACRO, default size is 32KB)(Phase2)
- BM can be viewed on LCDs
- Support sending BM’s log information to share memory when BM’s program is running and output of the log information under Linux.

- Both CPU cores can access the same memory address concurrently with the atomic operation guarantee.
- Both CPUs core can share DDR memory & OCM SRAM with cache coherence
- Cross-OS Spinlock/semaphore to protect shared resource in concurrent access
- BM interrupt latency is less than 0.5us and interrupt jitter is less than 0. 5us

# 2 Linux/BM Memory Maps&Attributes

## 2.1 Linux/BM Memory Map

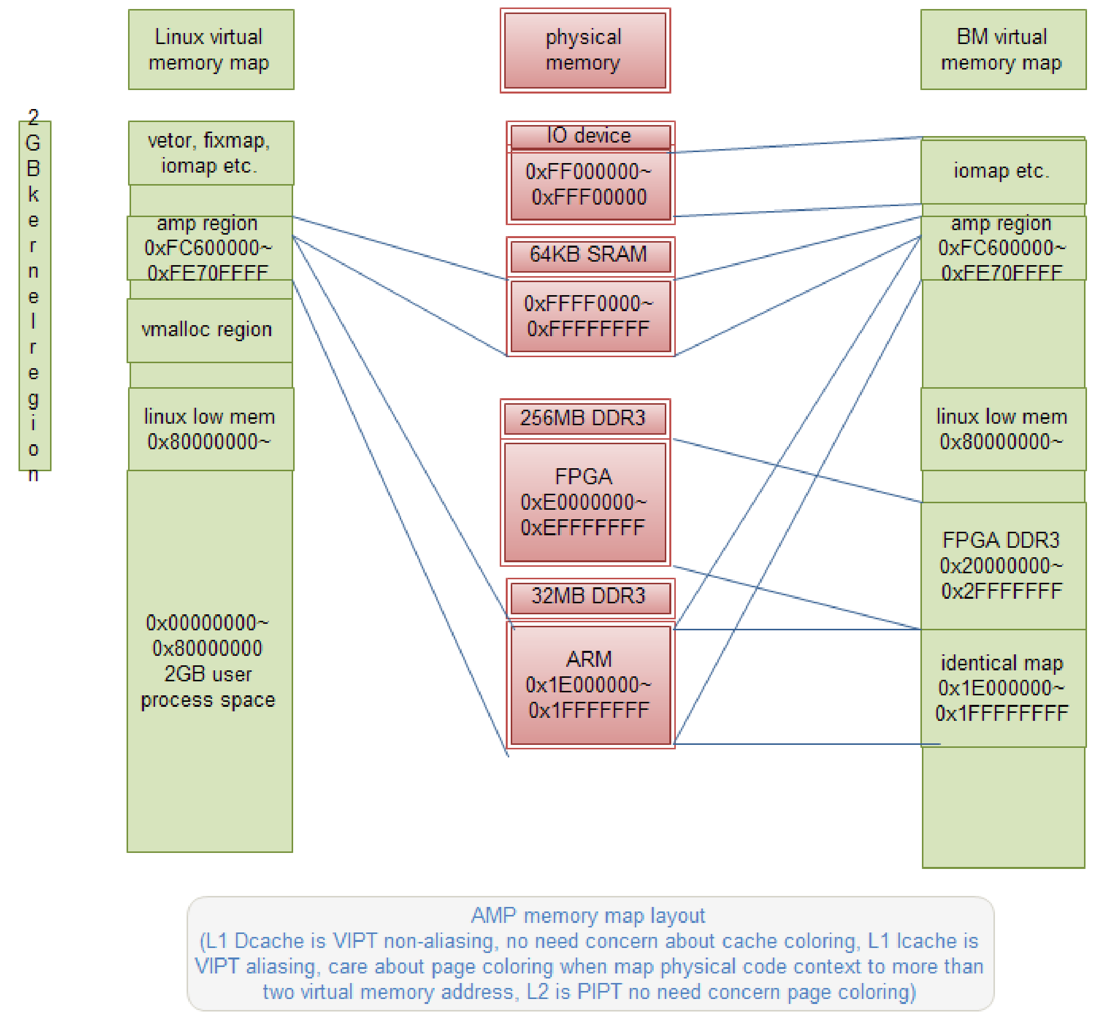

Linux virtual memory map
physical memory
BM virtual memory map
2GB kernel region
vetor, fixmap, iomap etc.
amp region 0xFC600000~ 0xFE70FFFF
vmalloc region
linux low mem 0x80000000~
0x00000000~ 0x80000000 2GB user process space
IO device
0xFF000000~ 0xFFF00000
64KB SRAM
0xFFFF0000~ 0xFFFFFFFF
256MB DDR3
FPGA 0xE0000000~ 0xEFFFFFFF
32MB DDR3
ARM 0x1E000000~ 0x1FFFFFFF
iomap etc.
amp region 0xFC600000~ 0xFE70FFFF
linux low mem 0x80000000~
FPGA DDR3 0x20000000~ 0x2FFFFFFF
identical map 0x1E000000~ 0x1FFFFFFF
AMP memory map layout
(L1 Dcache is VIPT non-aliasing, no need concern about cache coloring, L1 Icache is VIPT aliasing, care about page coloring when map physical code context to more than two virtual memory address, L2 is PIPT no need concern page coloring)

**Figure 2-1** Linux/BM Memory Map

## 2.2 Linux AMP Memmory Map&Attributes

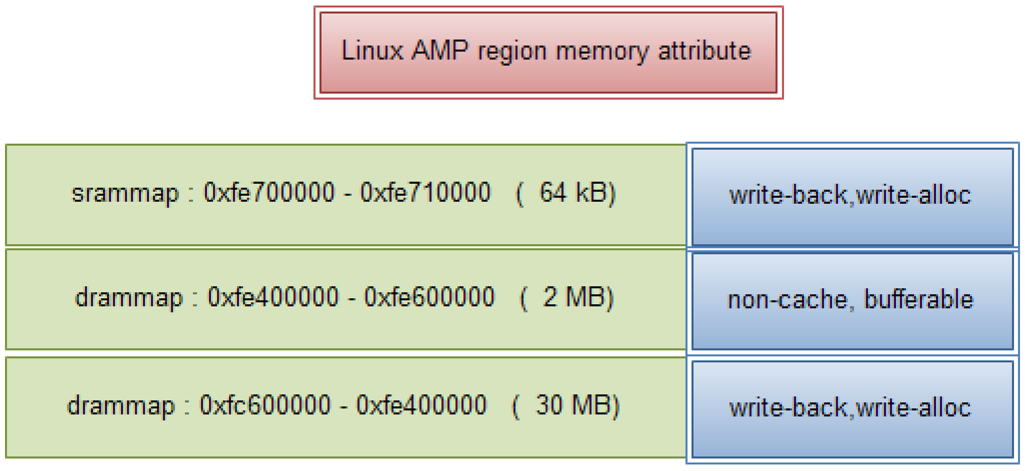
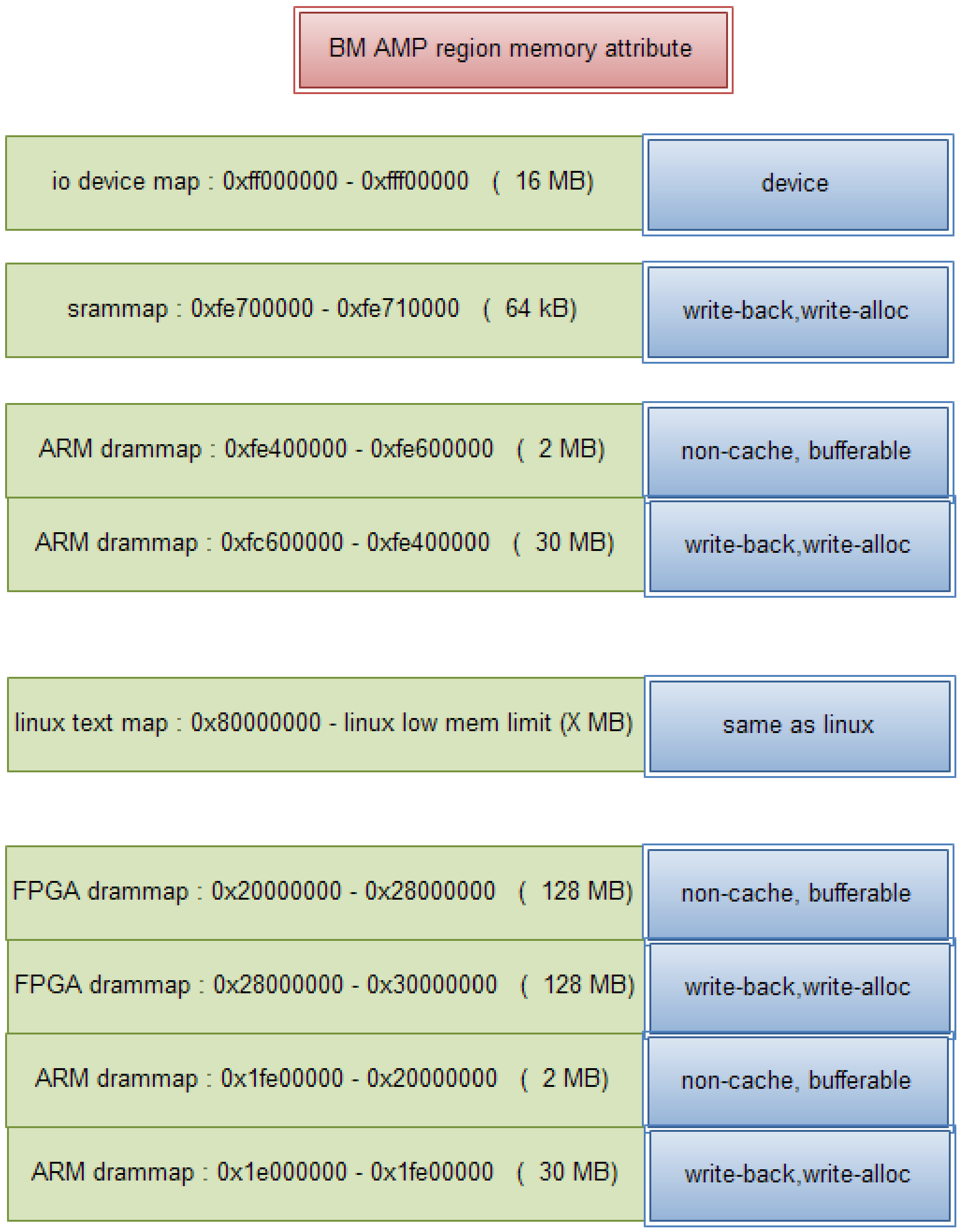

Linux AMP region memory attribute
srammap : 0xfe700000 - 0xfe710000 ( 64 kB)
write-back,write-alloc
drammap : 0xfe400000 - 0xfe600000 ( 2 MB)
non-cache, bufferable
drammap : 0xfc600000 - 0xfe400000 ( 30 MB)
write-back,write-alloc

**Figure 2-2** Linux AMP Memmory Map&Attributes

## 2.3 BM Memmory Map&Attributes



BM AMP region memory attribute

| | |
|---|---|
| io device map : 0xff000000 - 0xfff00000 ( 16 MB) | device |
| srammap : 0xfe700000 - 0xfe710000 ( 64 kB) | write-back,write-alloc |
| ARM drammap : 0xfe400000 - 0xfe600000 ( 2 MB) | non-cache, bufferable |
| ARM drammap : 0xfc600000 - 0xfe400000 ( 30 MB) | write-back,write-alloc |
| linux text map : 0x80000000 - linux low mem limit (X MB) | same as linux |
| FPGA drammap : 0x20000000 - 0x28000000 ( 128 MB) | non-cache, bufferable |
| FPGA drammap : 0x28000000 - 0x30000000 ( 128 MB) | write-back,write-alloc |
| ARM drammap : 0x1fe00000 - 0x20000000 ( 2 MB) | non-cache, bufferable |
| ARM drammap : 0x1e000000 - 0x1fe00000 ( 30 MB) | write-back,write-alloc |

**Figure 2-3** BM Memmory Map&Attributes

# 3 Boot of AMP

The following block diagram shows the booting process of AMP.

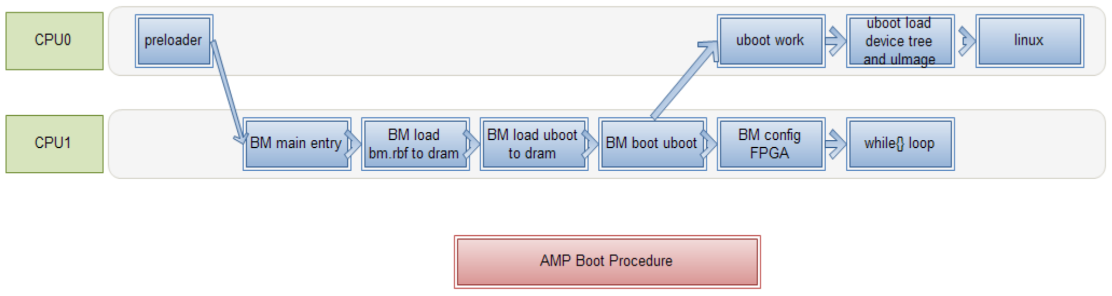


**Figure 3-1** Booting Process of AMP

## 3.1 Hardware Settings

The test of BM will be carried out on an Altera Cyclone V-based devkit board.The following settings for hardware are requiredbefore starting stest:

**Table 3-1** Hardware Settings

| Name | Value |
|---|---|
| Serial Terminal | 57600,8N1, no hardware flow control |
| MSEL[4...0] | 00000 (FPP16, no compression) |
| BOOTSEL[2..0] | 100(for sd/mmc flash memory boot)<br>111(for 3.0V qspi flash boot) |

## 3.2 Booting from QSPI/TF Card

### 3.2.1 Preparations

**Note:**

- Each instruction has been put a bullets "•" before it to prevent confusion caused by the long instructions that occupy more than one line in the context.
- Please note that there are SPACES put in the following instructions; Missing any SPACE will lead to failure when running an application.

1） Downloade Cross-Compiler from http://www.rocketboards.org/foswiki/Documentation/GsrdGitTrees and add the compiler into environment variable by executing the following instruction;

- **export PATH=/opt/altera-linux/linaro/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin/:$PATH**

2） Open the file build.h under board/altera/socfpga_cyclone5 and then define macros for QSPI flash or TF card booting according to your booting method;

**A. QSPI Booting:**

Define the macro as follows;

**Table 3-2** Macro Definition for QSPI Flash

```
#define CONFIG_PRELOADER_BOOT_FROM_QSPI          (1)
#define CONFIG_PRELOADER_BOOT_FROM_SDMMC         (0)
```

**B. TF Card Booting:**

Define the macro as follows;

**Table 3-3** Macro Definition for TF Card

```
#define CONFIG_PRELOADER_BOOT_FROM_QSPI          (0)
#define CONFIG_PRELOADER_BOOT_FROM_SDMMC         (1)
```

3） Execute the following instructions to compile u-boot;

- **make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- socfpga_cyclone5_config**
- **make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- all**

4） Execute the following instructions to compile kernel;

- **make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- LOADADDR=0x8000 menuconfig**
- **make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-**

**LOADADDR=0x8000 socfpga_amp_defconfig**

- **make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-**

  **LOADADDR=0x8000 uImage**

- **make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-**

  **LOADADDR=0x8000 dtbs**

### 3.2.2 Booting from QSPI Flash

The prcess of QSPI flash booting is started with BootROM loading preloader from QSPI flash. The preloader then takes over to load BM (Bare Metal) and release CPU1 RESET. After BM finishes loading RFB image from RAW partition of QSPI flash and the configuration of FPGA, preloader starts loading u-boot from QSPI flash right away, and then u-boot would finish the jobs of loading kernel and DTB, and running Linux system as well.

The following table lists thememory spaces where the codes should be positioned in QSPI flash.

**Table 3-4** Layout of QSPI Flash

| Codes | Memory Spaces |
|---|---|
| **Preloaders** | 0x00000000 - 256K (0x0 ~ 0x3ffff) 4 preloaders in total with each one 64KB |
| **ENV** | 0x40000 - 64KB |
| **FDT** | 0x50000 - 64KB |
| **u-boot** | 0x60000 - 256KB |
| **uImage** | 0xa0000 - 5MB Max |
| **BM** | 0x600000 - 128KB |
| **RBF** | 0x640000 - 12MB |
| **rootfs** | 0x1200000 - (128-18) 110MB |

According to the table shown above, the rootfs partition should be adjusted firstwithin DTS file of Linux kernel. You can program QSPI flash with the following instructions (marked in bold);

**Table 3-5** Programming QSPI Flash

```
SOCFPGA_CYCLONE5 # sf probe
SF: Detected N25Q00 with page size 64 KiB, total 128 MiB
```

```
SOCFPGA_CYCLONE5 # sf erase 0x0 0x2000000
SOCFPGA_CYCLONE5 # fatload mmc 0:1 0x4000000 preloader.bin
reading preloader.bin

65536 bytes read
SOCFPGA_CYCLONE5 # sf write 0x4000000 0x0 0x40000
SOCFPGA_CYCLONE5 # sf read 0x2000000 0x0   0x40000
SOCFPGA_CYCLONE5 # cmp.b 0x2000000 0x4000000 0x10000
byte at 0x02000000 (0xff) != byte at 0x04000000 (0x19)
Total of 0 byte(s) were the same
SOCFPGA_CYCLONE5 # sf erase 0x0 0x40000
SOCFPGA_CYCLONE5 # fatload mmc 0:1 0x4000000 preloader.bin
reading preloader.bin

65536 bytes read
SOCFPGA_CYCLONE5 # sf write 0x4000000 0x0 0x40000
SOCFPGA_CYCLONE5 # sf read 0x2000000 0x0   0x40000
SOCFPGA_CYCLONE5 # cmp.b 0x2000000 0x4000000 0x10000
Total of 65536 byte(s) were the same

SOCFPGA_CYCLONE5 # fatload mmc 0:1 0x4000000 u-boot.img
reading u-boot.img

241712 bytes read
SOCFPGA_CYCLONE5 # sf write 0x4000000 0x60000 0x40000
SOCFPGA_CYCLONE5 # sf read 0x2000000 0x60000   0x40000
SOCFPGA_CYCLONE5 # cmp.b 0x2000000 0x4000000 0x3B030
Total of 241712 byte(s) were the same
SOCFPGA_CYCLONE5 # fatload mmc 0:1 0x4000000 bm.bin
reading bm.bin

114688 bytes read
SOCFPGA_CYCLONE5 # sf write 0x4000000 0x600000 0x20000
SOCFPGA_CYCLONE5 # sf read 0x2000000 0x600000   0x20000
SOCFPGA_CYCLONE5 # cmp.b 0x2000000 0x4000000 0x1c000
Total of 114688 byte(s) were the same
SOCFPGA_CYCLONE5 # fatload mmc 0:1 0x4000000 uImage
reading uImage

2694296 bytes read
SOCFPGA_CYCLONE5 # sf write 0x4000000 0xa0000 0x400000
SOCFPGA_CYCLONE5 # sf read 0x2000000 0xa0000 0x291c98
SOCFPGA_CYCLONE5 # cmp.b 0x2000000 0x4000000 0x291c98
Total of 2694296 byte(s) were the same
```

```
SOCFPGA_CYCLONE5 # fatload mmc 0:1 0x4000000 socfpga.dtb
reading socfpga.dtb

7999 bytes read
SOCFPGA_CYCLONE5 # sf write 0x4000000 0x50000 0x10000
SOCFPGA_CYCLONE5 # sf read 0x2000000 0x50000 0x10000
SOCFPGA_CYCLONE5 # cmp.b 0x2000000 0x4000000 0x1f3f
Total of 7999 byte(s) were the same
SOCFPGA_CYCLONE5 # fatload mmc 0:1 0x4000000 jffs2.img
reading jffs2.img

11075584 bytes read
SOCFPGA_CYCLONE5 # sf write 0x4000000 0x1200000 0xa90000
SOCFPGA_CYCLONE5 # sf read 0x2000000 0x1200000 0xa90000
SOCFPGA_CYCLONE5 # cmp.b 0x2000000 0x4000000 0xa90000
Total of 11075584 byte(s) were the same

SOCFPGA_CYCLONE5 # fatload mmc 0:1 0x4000000 bm.rbf
reading bm.rbf

7007184 bytes read
SOCFPGA_CYCLONE5 # sf write 0x4000000 0x640000 0x700000
SOCFPGA_CYCLONE5 # sf read 0x2000000 0x640000 0x700000
SOCFPGA_CYCLONE5 # cmp.b 0x2000000 0x4000000 0x6AEBD0
Total of 7007184 byte(s) were the same
```

It would take 32ms approximately from the moment of loading preloader into the on-chip RAM to the first instruction being run by BM. The time is measured by using OSC timer1, while timer0 is used to measure the time needed for running preloader and u-boot. The OSC timer is working at a fixed frequency of 25MHz.

**Note:**

- The QSPI driver is reused by preloader in the BM.
- Currently the QSPI driver can work properly as it is reading, but could be unstable occasionally when wrting, and therefore an unexpected failure may occur when programming QSPI flash. Check QSPI flash image when encountering the issue, or erase the image and program the flash again.

### 3.2.3 Booting from TF Card

The prcess of TF card booting is started with BootROM loading preloader from RAW partition of TF card. The preloader then takes over to load BM (Bare Metal) and release CPU1 RESET. After BM finishes loading RFB image from TF card and the configuration of FPGA, preloader starts loading u-boot from RAW partition of TF card, and then u-boot would finish the jobs of loading kernel and DTB from FAT partition of TF card, and running Linux system as well.

The following table lists the memory spaces where the codes are positioned in the RAW partition of a TF card.

**Table 3-6** Layout of RAM Partition

| Codes | Absolute Address | Relative Address to RAW Partition | Size |
|---|---|---|---|
| **Preloaders** | 0x100000 | offset is 0x00000 | 256KB(64KB/per preloader) |
| **u-boot.img** | 0x140000 | offset is 0x40000 | 256KB |
| **bm.bin** | 0x200000 | offset is 0x100000 | 1MB |
| **rbf.bin** | 0x300000 | offset is 0x200000 | 14MB |

1） Use the following bash script to generate an image for TF card. (the script comes from Altera's make_sdimage.sh)

**Table 3-7** Bash Script

```
sudo ./make_sdimage_bm.sh -k uImage,socfpga.dtb
                          -p preloader.bin
                          -b u-boot.img
                          -a bm.bin
                          -f bm.rbf
                          -r your local directory/ rootfs/
                          -o sd_image_bm.bin
```

where:

**-k:** uImage and dtb

**-p:** preloader

**-b:** u-boot

**-a:** bm

**-f:** rbf

**-r:** the rootfs which is generated from the yocto

**-o:** the output image name

For the information of how to upate images in a TF card, please visit http://www.rocketboards.org/foswiki/Documentation/GSRDBootLinuxSd.

2） Execute the following instructions to update preloader, u-boot.img, bm, bmrbf respetively;

- **sudo dd if=preloader.bin of=/dev/sdb3 bs=64k seek=0;sudo sync**
- **sudo dd if=u-boot.img of=/dev/sdb3 bs=64k seek=4;sudo sync**
- **sudo dd if=bm.bin of=/dev/sdb3 bs=1M seek=1;sudo sync**
- **sudo dd if=bm.rbf of=/dev/sdb3 bs=2M seek=1;sudo sync**

**Note:**

📖 You need to program an image into the TF card first before you can update the files mentioned above.

3） The test results are shown in the following table;

**Table 3-8** Test Results

| | |
|---|---|
| Kingston class 4 4GB: | |
| boot time: 46-46ms | load bm time(inlcude init sd card): 23ms |
| Sony class 10 16GB: | |
| boot time: 44ms | load bm time(include init sd card): 20ms |

The SD driver in the BM reused the SD driver of preloader. BM will choose either TF card or QSPI flash as the source to load RBF for FPGA according to the pin setting of BOOTSEL[2:0].

# 4 BM Log under Linux

Please execute the following instruction in Linux console to view BM log;

- **cat /sys/class/amp/amp/bm_log**

The information in Linux console is shown below;

**Table 4-1** BM Log

```
logindex = 1900
start bm......6000
i am from fpga ddr ram
i am from fpga sram
L1 L2 cache enabled, SCU enabled~~~
BM L2 Lock ways status:D:0x0000005f I:0x0000005f

________________________________________________________________________
____________
Now the BM CPU would suffer more than 16000 interrupts/second(10000i/s from
FPGA_IRQ0 req(100us),more than 6000i/s from IPI(Inner Process Interrupt), and
process several Gbps data per second, they are all concurrently, very intensive
load for BM CPU core!
________________________________________________________________________
______________

---------------------------
BM spinlock test:3996862
---------------------------

-------------------------------------------------------------------------------------
interrupt latency test method 1(use private timer)
max interrupt latency jitter: 260 ns
average interrupt latency jitter: 20 ns
max interrupt latency: 350 ns(use private timer@CPU core cluster)
average interrupt latency: 110 ns(use private timer@CPU core cluster)
------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------
fpga send 100002 times irq req to arm, arm ack 100002 times, lost irq 0 times
--------------------------------------------------------------------------------------------
DMA test memory(0x1ff04000) to memory(0x1ff84000) is ok!
DMA test done
--------------------------------------------
```

```

DMA test memory(0x1e204000) to memory(0x1e208000) is ok!
DMA test acp done
-----------------------------------------------

DMA test memory(0x1ff04000) to memory(0xe8004000) is ok!
dma_ARMmem2FPGAmem test done
-----------------------------------------------

DMA test memory(0xe0004000) to memory(0x1e304000) is ok!
dma_FPGAmem2ARMmem_use_acp test done
-----------------------------------------------

root@socfpga_cyclone5:~#
```

# 5 Test of AMP Reference Design

### 5.1.1 Test with Linux Preempt Enabled

Please execute the following instruction to begin the test with Linux preempt enabled;

- **echo test > /sys/class/amp/amp/amp_test**

The information in Linux console is shown below;

**Table 5-1** Test Information

```
roamp test begin
ot@socfpga_cyclone5:~#
DRAM share test finished(test DRAM share access and cache
coherence)
SRAM share test finished(test SRAM share access and cache
coherence)
Linux spinlock test:4000000
waiting BM finish interrupt latency test and DMA test etc.
BM finish BM finish interrupt latency test and DMA test
use cat /sys/class/amp/amp/bm_log to check bmlog

root@socfpga_cyclone5:~#
```

### 5.1.2 Test with Linux Preempt Disabled

Please execute the following instruction to begin the test with Linux preempt enabled;

- **echo test_wpd > /sys/class/amp/amp/amp_test**

# 6 Test Options for AMP Reference Design

1） Booting two OS/BM on two CPU core;

2） BM boot time;

OSC1 timercan be used to measure the boot time. The timer can be started in preloader start.S entry and stopped in BM main entry.

3） BM config FPGA;

BM config FPGA usesRBF(FPP16 format) config image.

4） Memory pollingon two cpu core;

Please refer to DRAM share and cache coherence test.

5） IPI(Inter-Processor Interrupt);

Please refer to DRAM share and cache coherence test;

6） DRAM share and cache coherence test;

**Table 6-1** RAM Share/Cache Coherence Test

| Steps | For Linux | For BM |
|---|---|---|
| **1** | CPU0 writes DATA_PATTERN1 to 64KB DRAM_BUF | CPU1 is idle or doing it’s private work |
| **2** | CPU0 sends IPI to CPU1 to start verification | CPU1 verifies 64KB DRAM_BUF of DATA_PATTERN1 |
| **3** | CPU0 is idle or doing it’s private work | CPU1 writes DATA_PATTERN3 to 64KB DRAM_BUF |
| **4** | CPU0 verifies 64KB DRAM_BUF of DATA_PATTERN3 | CPU1 sends IPI to CPU0 to start verification |
| **5** | Goto step 1) and repeat 40K times | |

7） SRAM share and cache coherence test;

The process is as same as DRAM test.

8） Printk;

Just like linux printk, the function printk will disable interrupt to access textbuf; Try to use bmlog instead of using printk in real time critical process.

**9**） Semaphore(test case have been embedded in printk);

Semaphore is used to protect global resource/data struct shared by Linux and BM.Refer to linux semaphore up/down/down_trylock primitive.

**10**） Bmlog;

Dump debug/log message to logbuf.Use command **cat /sys/class/amp/amp/bm_log** view logbuf.

**11**） Spinlock;

Spinlock is used to protect global resource/data struct share by linux and bm.Refer to linux spinlock spinlock_lock/spinlock_unlock/etc primitive.

**12**） Interrupt jitter &interrupt latency (using private timer in Cortex-A9 BM core);

The following block diagrams illustrate the testing processes of interrupt jitter;

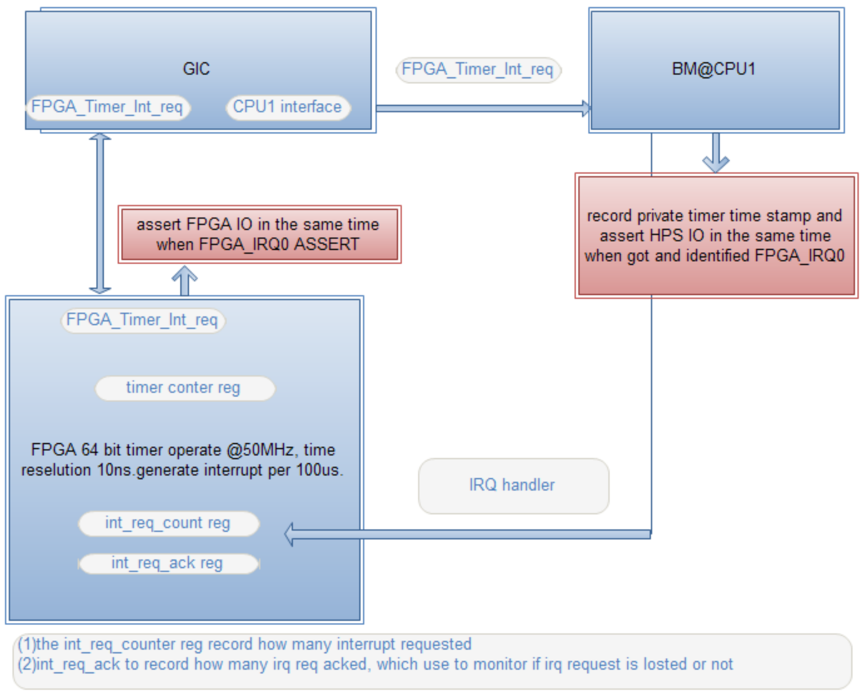

GIC
FPGA_Timer_Int_req
CPU1 interface
FPGA_Timer_Int_req
BM@CPU1
assert FPGA IO in the same time when FPGA_IRQ0 ASSERT
record private timer time stamp and assert HPS IO in the same time when got and identified FPGA_IRQ0
FPGA_Timer_Int_req
timer conter reg
FPGA 64 bit timer operate @50MHz, time reselution 10ns.generate interrupt per 100us.
IRQ handler
int_req_count reg
int_req_ack reg
(1)the int_req_counter reg record how many interrupt requested
(2)int_req_ack to record how many irq req acked, which use to monitor if irq request is losted or not

**Figure 6-1** Testing Process of Interrupt Jitter

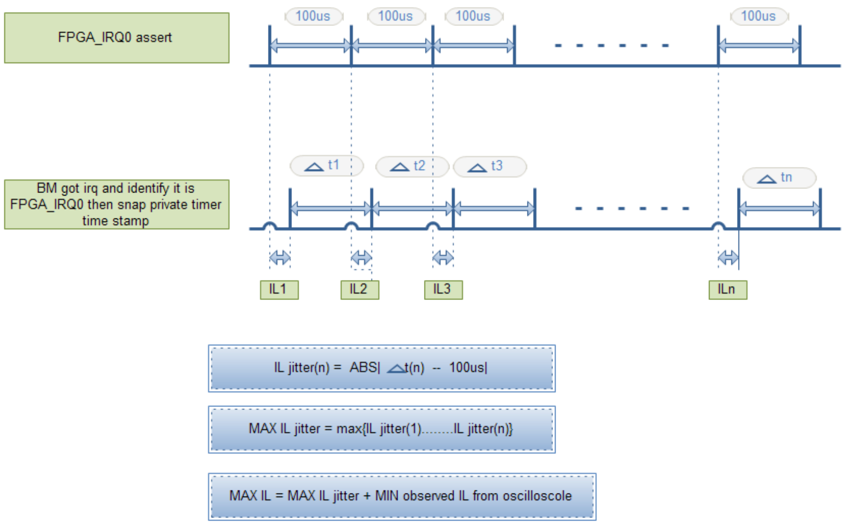

FPGA_IRQ0 assert
100us
100us
100us
100us
BM got irq and identify it is FPGA_IRQ0 then snap private timer time stamp
t1
t2
t3
tn
IL1
IL2
IL3
ILn
IL jitter(n) = ABS| t(n) -- 100us|
MAX IL jitter = max{IL jitter(1)........IL jitter(n)}
MAX IL = MAX IL jitter + MIN observed IL from oscilloscole

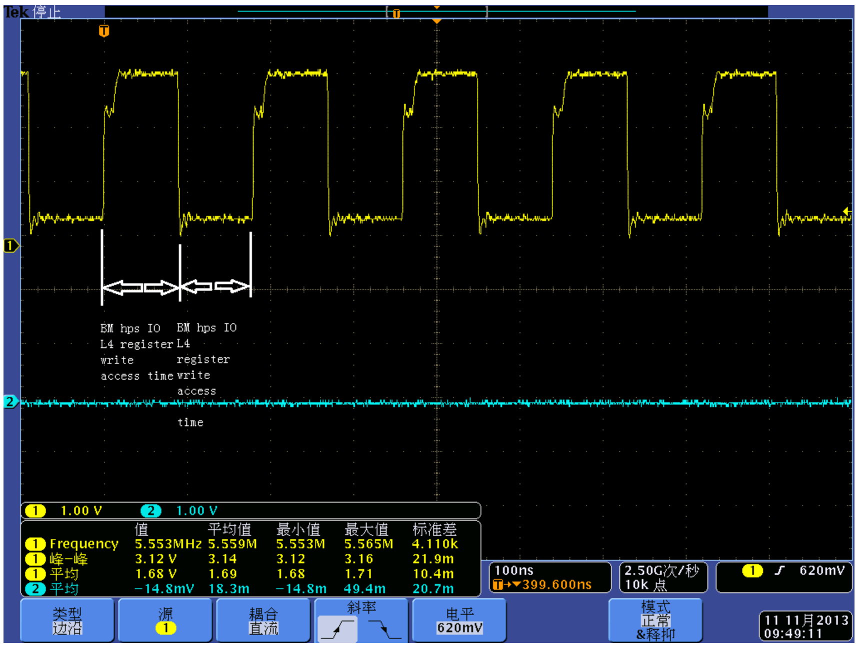
**Figure 6-2** Testing Process of IL Time

In order to get the observed IL from oscilloscope, we need to test the BM hps IO write access time first. The L4 IO write access code is shown below;

**Table 6-2** L4 Code for Testing 1

```
while(1)
{
    *(volatile u32 *)0xff709000 = (0x1<<12);
    *(volatile u32 *)0xff709000 = ~(0x1<<12);
}
```

The following figure is an oscilloscope screenshot which shows IO L4 write access time;



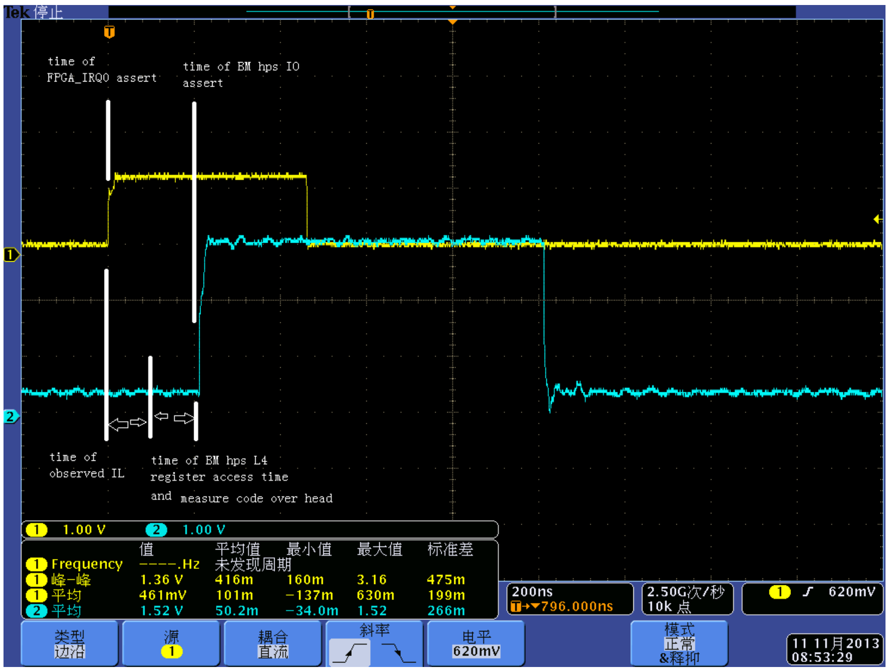
| | 值 | 平均值 | 最小值 | 最大值 | 标准差 |
|---|---|---|---|---|---|
| 1 Frequency | 5.553MHz | 5.559M | 5.553M | 5.565M | 4.110k |
| 1 峰-峰 | 3.12 V | 3.14 | 3.12 | 3.16 | 21.9m |
| 1 平均 | 1.68 V | 1.69 | 1.68 | 1.71 | 10.4m |
| 2 平均 | −14.8mV | 18.3m | −14.8m | 49.4m | 20.7m |

**Figure 6-3** Oscilloscope Screenshot 1

The IO electronic delay time and cpu ldr/str time to register/cache can be ignored since they are much less than L4 access time.We can learn from the screenshot that IO L4 write access time is 90 ns.

Then BM exception handler asserts hps IO when BM gets FPGA_IRQ0, the delta time from FPGA IO assert (in the same time FPGA_IRQ0 assert to BM) to hps IO assert minus the BM hps L4 write access time and the measure code overhead is the IL time.

The following figure is an oscilloscope screenshot which shows observed IL time;



| | 值 | 平均值 | 最小值 | 最大值 | 标准差 |
|---|---|---|---|---|---|
| 1 Frequency | ----.Hz | 未发现周期 | | | |
| 1 峰-峰 | 1.36 V | 416m | 160m | 3.16 | 475m |
| 1 平均 | 461mV | 101m | −137m | 630m | 199m |
| 2 平均 | 1.52 V | 50.2m | −34.0m | 1.52 | 266m |

**Figure 6-4** Oscilloscope Screenshot 2

So observed IL time = delta t from FPGA IO assert to hps IO assert - BM hps L4 write access time - the measure code overhead = 200ns/grid*1grid – 90ns – about20ns = 90ns.as figure 5-4

MAX IL = MAX IL jitter + 90ns.

**13**） DMA xfer;

Transfer data from non-cacheable, bufferable memory to non-cacheable, bufferable memory.

**14**） ACP xfer;

Transfer data from ARM cacheable, bufferable memory to ARM cacheable, bufferable memory.

Transfer data from FPGA cacheable, bufferable memory to ARM cacheable, bufferable memory.

**15**） Testing data process speed of two cortex-a9 core;

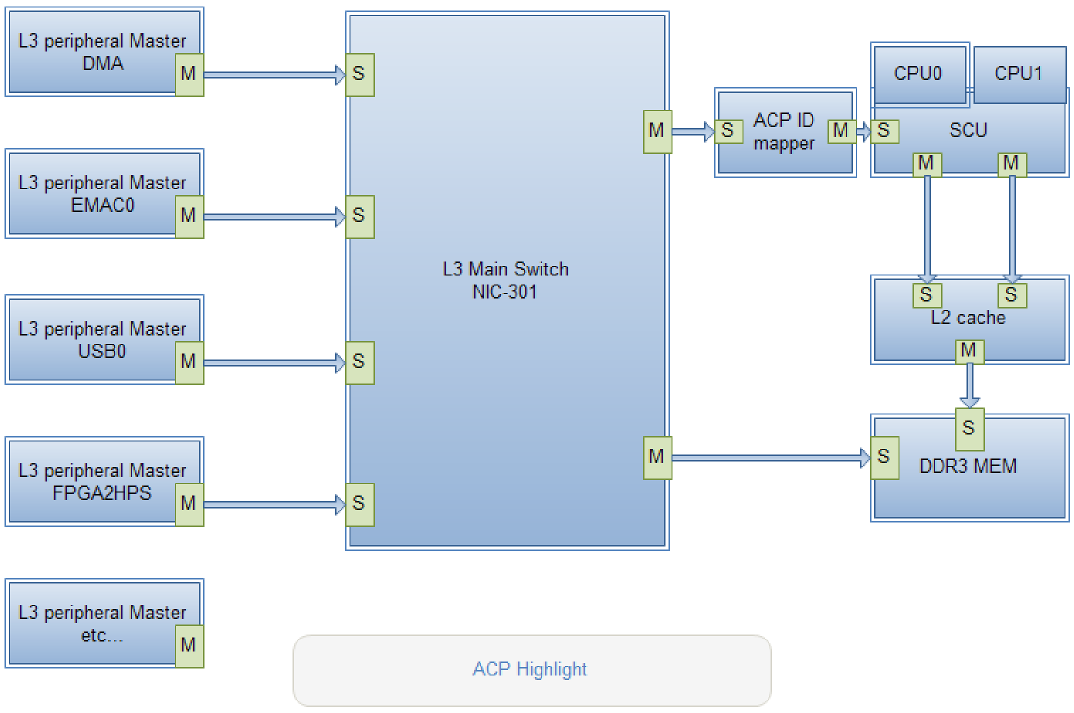
Please refer to BM main function comment.

**16**） Measuring BM while{} loop period & instruction/cycle

Please refer to BM main function comment.

# 7 ACP programming guide

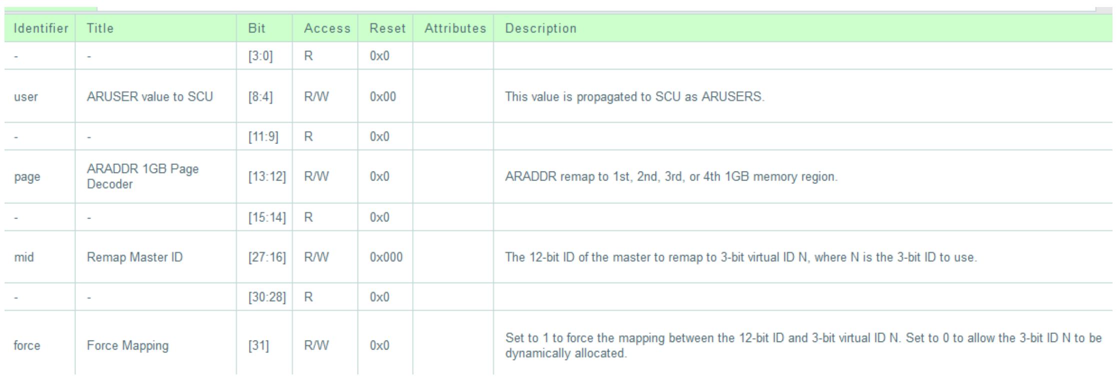
Before we start describe the ACP/L1/L2 cache coherence, let's take a look at the ACP/SCU/L1/L2 highlight figure.



In this figure, each cpu issue transaction with virtual/modified virtual address which combine with the translation table to determine how the cache maintenance behavior are when thus transactions travel to L1 cache & L2 cache. But every L3 master(DMA, EMAC, FPGA, etc…) is working in the physical address mode, how could the L3 master know about cache maintenance information when it issue L3 transactions, the solution is use sideband signal as a complementary AXI transaction signal, then ACP ID mapper use this sideband signal to determine thus L3 master transaction cache maintenance behavior when travel across SCU.

We would take the FPGA DDR3 memory to ARM DDR3 memory DMA transfer ACP configuration example as ACP programming guide.

ACP ID Mapper Registers



| Identifier | Title | Bit | Access | Reset | Attributes | Description |
|---|---|---|---|---|---|---|
| - | - | [3:0] | R | 0x0 | | |
| user | ARUSER value to SCU | [8:4] | R/W | 0x00 | | This value is propagated to SCU as ARUSERS. |
| - | - | [11:9] | R | 0x0 | | |
| page | ARADDR 1GB Page Decoder | [13:12] | R/W | 0x0 | | ARADDR remap to 1st, 2nd, 3rd, or 4th 1GB memory region. |
| - | - | [15:14] | R | 0x0 | | |
| mid | Remap Master ID | [27:16] | R/W | 0x000 | | The 12-bit ID of the master to remap to 3-bit virtual ID N, where N is the 3-bit ID to use. |
| - | - | [30:28] | R | 0x0 | | |
| force | Force Mapping | [31] | R/W | 0x0 | | Set to 1 to force the mapping between the 12-bit ID and 3-bit virtual ID N. Set to 0 to allow the 3-bit ID N to be dynamically allocated. |

Example: FPGA DDR3 memory to ARM DDR3 memory DMA transfer
FPGA DDR3 memory@0xE0000000 ----ACP transfer to--→ ARM DDR3 memory@0x1E300000

1. Setting DMA master reader(source) property
   Step 1: confirm the source ACP window, cause 0xE0000000 is locate at 4th GB of 4GB address space. So the "ARADDR 1GB Page Decoder" bits field is 0x3

   Step 2: confirm the source cache attributes, cause physical memory at 0xE0000000 is remap as wbwa sharable, through the table below we could determine that the "ARUSER value to SCU" bits field is 0x1F

```
sideband signal     AXUSER[4..0]:     [4:1] inner attributes:
                                            b0000 = Strongly-ordered
                                            b0001 = Device
                                            b0011 = Normal Memory Non-Cacheable
                                            b0110 = Write-Through
                                            b0111 = Write-Back no Write Allocate
                                            b1111 = Write-Back Write Allocate
                                            [0] shared.
```

   Step 3: confirm master id bits filed, cause we use DMA channel 0 do the dma job, through the micro below we can determine the "Remap Master ID" bits field.

```
12bit master id

 Interconnect Master                    | ID
 ---------------------------------------------------
 L2M0                                              | 12'b0xxxxxxxx010
 DMA                                               | 12'b00000xxxx001
 EMAC0                                             | 12'b10000xxxx001
 EMAC1                                             | 12'b10000xxxx010
 USB0                                              | 12'b100000000011
  NAND                                             | 12'b1xxxxxxxx100
 TMC                                               | 12'b100000000000
  DAP                                              | 12'b000000000100
 SDMMC                                             | 12'b100000000101
 FPGA2HPS bridge                                   | 12'b0xxxxxxxx000
  USB1                                             | 12'b100000000110
  ---------------------------------------------------
where:
          for dma: when the   DMA channel thread accesses the DMA interface, xxxx=0yyy,
where yyy is chnannel number
                        When the DMA manager thread accesses the DMA interface, xxxx=1000
= number of channels
          Please refer to the newest the cyclone5_handbook.pdf.


#define ACP_ID_DMA_CHANNEL_THREAD(x)                        (0x1 | ((x & 0xf) <<
3))
```

Step 4: we use fix master ID here, so we make the "Force Mapping" bit field as 1.

2. Setting DMA master writer(dest) property
   Step 1: confirm the dest ACP window, cause 0x1E300000 is locate at 1st GB of 4GB address space. So the "ARADDR 1GB Page Decoder" bits field is 0x0

   Step 2: confirm the dest cache attributes, cause physical memory at 0x1E300000 is remap as wbwa sharable, through the table below we could determine that the "ARUSER value to SCU" bits field is 0x1F

```
sideband signal      AXUSER[4..0]:       [4:1] inner attributes:
                                              b0000 = Strongly-ordered
                                              b0001 = Device
                                              b0011 = Normal Memory Non-Cacheable
                                              b0110 = Write-Through
                                              b0111 = Write-Back no Write Allocate
```

```
                                                    b1111 = Write-Back Write Allocate
                                                    [0] shared.
```

Step 3: confirm master id bits filed, cause we use DMA channel 0 do the dma job, through the micro below we can determine the “Remap Master ID” bits field.

```
12bit master id

 Interconnect Master                    | ID
 ----------------------------------------------------
 L2M0                                   | 12'b0xxxxxxxx010
 DMA                                    | 12'b00000xxxx001
 EMAC0                                  | 12'b10000xxxx001
 EMAC1                                  | 12'b10000xxxx010
 USB0                                   | 12'b100000000011
  NAND                                  | 12'b1xxxxxxxx100
 TMC                                    | 12'b100000000000
  DAP                                   | 12'b000000000100
 SDMMC                                  | 12'b100000000101
 FPGA2HPS bridge                        | 12'b0xxxxxxxx000
  USB1                                  | 12'b100000000110
  ----------------------------------------------------
where:
        for dma: when the   DMA channel thread accesses the DMA interface, xxxx=0yyy,
where yyy is chnannel number
                    When the DMA manager thread accesses the DMA interface, xxxx=1000
= number of channels
        Please refer to the newest the cyclone5_handbook.pdf.


#define ACP_ID_DMA_CHANNEL_THREAD(x)                    (0x1 | ((x & 0xf) <<
3))
```

Step 4: we use fix master ID here, so we make the “Force Mapping” bit field as 1.

And here is the configuration code:

```
/* Init ACP ID Mapper register */
```

```
static void dma_use_acp_init_fpga(ALT_DMA_CHANNEL_t channel_id)
{
        volatile u32 * reg;
        u32 val;

        val = (ACP_ID_DMA_CHANNEL_THREAD(channel_id) << 16) | (0x1 << 31)| (0x1f << 4) |
(0x3<<12);
        reg = ACP_REG_ID_MAP(ACP_REG_ID_3_RD);
        *reg = val;

        val = (ACP_ID_DMA_CHANNEL_THREAD(channel_id) << 16) | (0x1 << 31)|(0x1f << 4);
        reg = ACP_REG_ID_MAP(ACP_REG_ID_3_WR);
        *reg = val;

}
```

# 8 Trigger SGI0~7 in user space

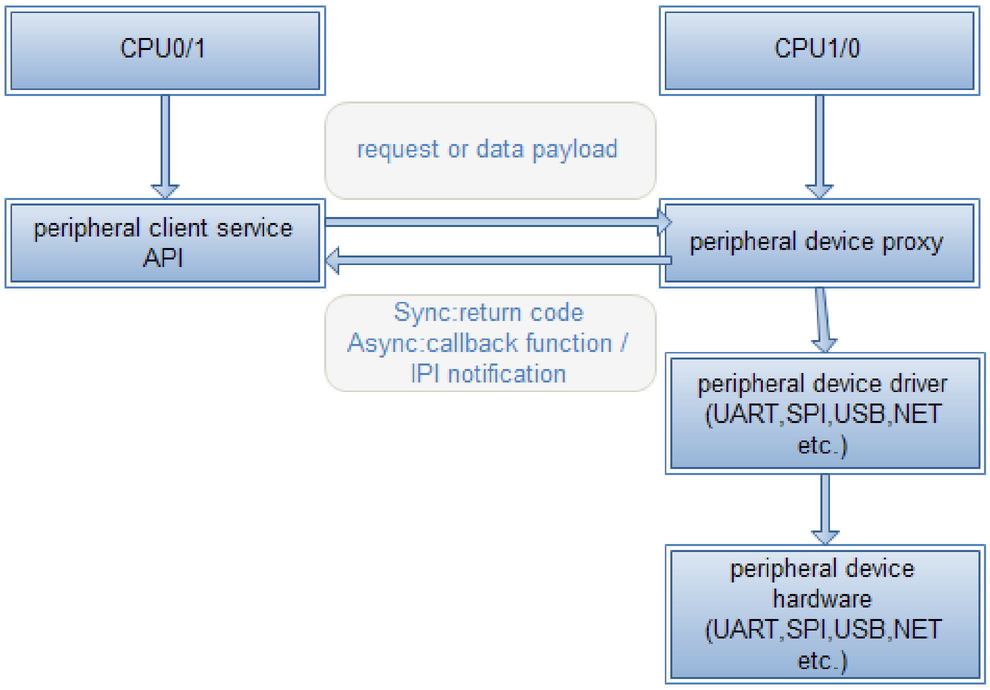
Cortex-A9 MPCORE support 16 SGI interrupt, the AMP reference design have export the SGI0~SGI7 total 8 SGIs to sys system user space, use the command below to trigger the SGI to BM core and verify bm log.

```
root@socfpga_cyclone5:~# echo 0 > /sysass/amp/amp/sgi_trigger
root@socfpga_cyclone5:~# echo 2 > /sysass/amp/amp/sgi_trigger
root@socfpga_cyclone5:~# echo 4 > /sysass/amp/amp/sgi_trigger
root@socfpga_cyclone5:~# echo 6 > /sysass/amp/amp/sgi_trigger
root@socfpga_cyclone5:~# echo 8 > /sysass/amp/amp/sgi_trigger
invalid sgi number
root@socfpga_cyclone5:~# echo 1 > /sysass/amp/amp/sgi_trigger
root@socfpga_cyclone5:~# echo 33 > /sysass/amp/amp/sgi_trigger
invalid sgi number
root@socfpga_cyclone5:~# echo 3 > /sysass/amp/amp/sgi_trigger
root@socfpga_cyclone5:~# echo 5 > /sysass/amp/amp/sgi_trigger
root@socfpga_cyclone5:~# echo 7 > /sysass/amp/amp/sgi_trigger
root@socfpga_cyclone5:~# cat /sysass/amp/amp/bm_log
logindex = 473
start bm......6000
load fpga rbf is ok!
start[0xffffff9f],end[ffed4d01]
, bm_load_ns = 49014960(ns), bm_load_ms= 49(ms)
real loading bm time duration is 20(ms)(including qspi/sd init)
i am from fpga ddr ram
i am from fpga sram
L1 L2 cache enabled, SCU enabled~~~
BM L2 Lock ways status:D:0x0000005f I:0x0000005f
BM SGI#0 triggered!
BM SGI#2 triggered!
BM SGI#4 triggered!
BM SGI#6 triggered!
BM SGI#1 triggered!
BM SGI#3 triggered!
BM SGI#5 triggered!
BM SGI#7 triggered!
root@socfpga_cyclone5:~#
```

# 9 AMP APP Development tips

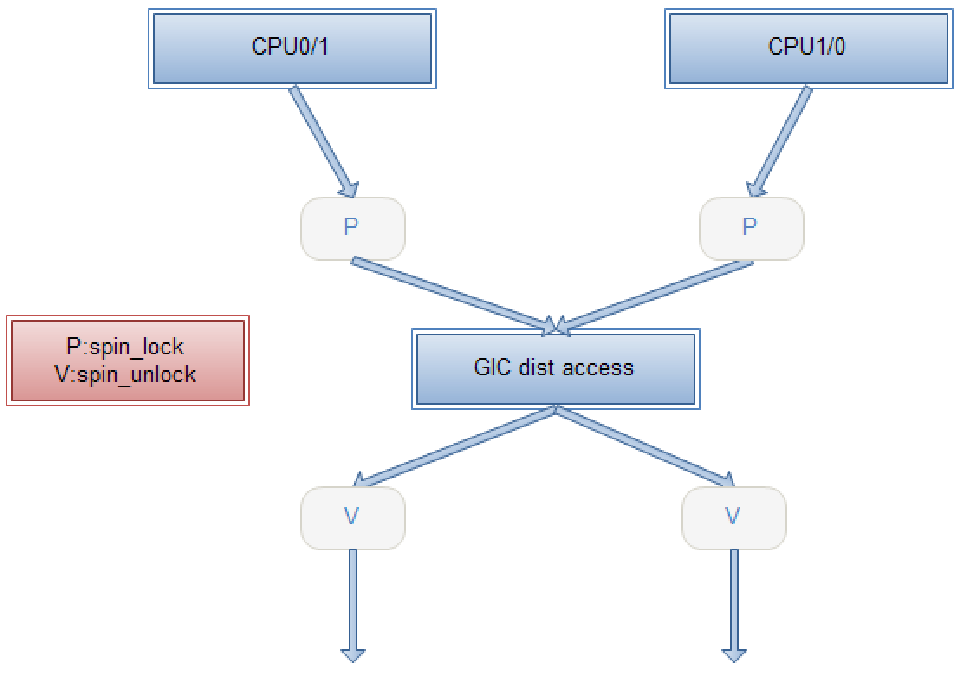
1. Peripheral device share by Linux & BM

   1) Proxy


CPU0/1
CPU1/0
request or data payload
peripheral client service API
peripheral device proxy
Sync:return code
Async:callback function / IPI notification
peripheral device driver (UART,SPI,USB,NET etc.)
peripheral device hardware (UART,SPI,USB,NET etc.)

   2) P/V primitive access

      take GIC distributor as example, GIC cpu interface is private, no need to protect, GIC distributor some registers is base on set/clr access which means is atomic also no need to protect.



2. you need save/restore VFP/NEON dedicated data & status register in exception handler if you use VFP/NEON in your ISR handler, use VFP/NEON only in synchronous context is no need to do this routine.

3. Linux & BM Asynchronous Communication

1) IPI(Inner Processer Interrupt)

2) Call back function

4. Linux & BM Synchronous Communication

1 )spinlock (both Linux&BM available)

2 )semaphore (both Linux&BM available, BM do not call the API would have change to block, invoke the try version instead)

3) mutex (both Linux&BM available, BM do not call the API would have change to block, invoke the try version instead)

4 )rwlock (both Linux&BM available, BM do not call the API would have change to block, invoke the try version instead)

5) seqlock (both Linux&BM available, for seqlock writer BM do not call the API would have change to block, invoke the try version instead, reader's behavior is like spinlock don't worry to be block)

6 )RCU (Not suitable for BM cause which have no scheduler)

7 )share memory

5.How to Zero Copy to Access Linux&BM share kernel memory in usespace.

Please refer to linux user space application amp_app.c.

504365056 (dec) = 0x1e100000(hex)

```
root@socfpga_cyclone5:~# mount /dev/mmcblk0p1 /media/card/

root@socfpga_cyclone5:~# /media/card/amp_app

usage: #amp_app /dev/amp phy_addr size

root@socfpga_cyclone5:~# /media/card/amp_app /dev/amp 504365056 4096
```

# 10 AMP Device Driver & Changes of Linux Kernel and u-boot

The AMP device driver **amp.c**is located under linux_kernel_top_dir/driver/char/;

The following table lists all the files that have been changed in Linux kernel;

**Table 10-1** Changes in Linux Kernel

| No. | Location |
|---|---|
| **1** | Arch/arm/mach-socfpga/socfpga.c (memblock reserved for BM) |
| **2** | Arch/arm/command/gic.c (don't make any change to BM-relatedIRQs property) |
| **3** | Arch/arm/kernel/smp.c (exporting kernel function to bm etc.) |
| **4** | Arch/arm/mm/mmu.c (creating fixed mapping for amp) |
| **5** | Arch/arm/include/asm/pgtable.h (adjusting vmalloc region and allocating reserved region to amp) |
| **6** | Arch/arm/include/asm/amp_config.h (defining amp parameters) |
| **7** | Arch/arm/mm/cache-l2x0.c (locking l2ways by master, and making Linux don'ttouch to BM ways) |

We have updated the qspi driver and sd mmc driver under u-boot-2013.01.01, and modified the spl for loading bm.

**Table 10-2** Changes in u-boot

| No. | Location |
|---|---|
| **1** | Board/altera/socfpga_cyclone5/build.h(spl for sd/qspi, and remove the serial support for spl) |
| **2** | Include/configs/socfpga_cyclone5.h(for qspi and sd) |
| **3** | Arch/arm/include/ams/arch-socfpga/amp_config.h(exportingspl function to bm etc.) |
| **4** | Arch/arm/cpu/armv7/start.S (for boot start time stamp) |
| **5** | Arch/arm/cpu/armv7/osc_timer1.S (the osc timer1 for calculating boot time for bm) |
| **6** | Common/spl/spl.c (load bm for sd and qspi) |
| **7** | Board/altera/socfpga_cyclone5/socfpga_cyclone5.c(boot bm) |
| **8** | Arch/arm/cpu/armv7/socfpga/clock_manager.c(delay using udelay function about 7us,not 7ms) |
| **9** | Arch/arm/cpu/armv7/socfpga/spl.c(don't reset the osc timer1) |
| **10** | Drivers/mtd/spi/spi_spl_load.c(load bm from the qspi) |
| **11** | Drivers/spi/cadence_qspi.c,cadence_qspi_apb.c,Drivers/mtd/spi/spi_flash.c(update for qspi driver) |
| **12** | Drivers/mmc/spl_mmc.c(load bm from the sd card) |
| **13** | Arch/arm/include/asm/arch-socfgpa(dwmmc.h),Drivers/mmc/altera_dw_mmc.c,dw_mmc.c(upate for sd/mmc driver) |

# Contact Information

**Technical Support**

Telephone Number: +86-755-25635626-872/875/897

Email Address: support@embedinfo.com

**Sales Information**

Telephone Number: +86-755-25635626-860/861/862

Fax Number: +86-755-25616057

Email Address: market@embedinfo.com

**Company Information**

Company Website: http://www.embest-tech.com

Company Address: Tower B 4/F, Shanshui Building, Nanshan Yungu Innovation Industry Park, Liuxian Ave. No. 1183, Nanshan District, Shenzhen, Guangdong, China (518055)