IPC MMU fault debug

Contents

Debugging MMU faults when using IPC 3.x

Problem Description

MMU faults may occur on a remote slave core during execution of an IPC-based program due to bugs that were inadvertently introduced. They are encountered on systems where an MMU is available on the slave to guard it from accessing invalid memory. On systems were the host is running a high-level OS such as QNX or Linux, when such an access occurs, an interrupt is sent to the host processor, which would perform recovery. As of IPC 3.21, the recovery process consists of resetting, reloading and restarting all slave cores. This fail-safe ensures that both host-to-slave and slave-to-slave communication, if any, can restart from a clean slate. This could be handy in a production system where the system can reboot itself when a bug is encountered.

However, even when recovery is successful, this is still a bug, and a program should not access invalid memory locations.

This topic describes techniques on how to debug such an MMU fault.

> **NOTE**
>
> Some CPUs include an additional MMU called the 'AMMU'. In this topic, we focus on faults at the level of the 'IOMMU', which is programmed externally by the host and would trigger an interrupt that initiates the recovery mechanism upon the occurrence of a fault.

Prerequisites

- Development board setup with JTAG access thru CCS 5.x or above
- Familiarity with connecting to slave cores on your device using CCS for debug purposes
- Familiarity with rebuilding IPC

Approach

Identify the fault address

When debugging an MMU fault, one should start by identifying the fault address. IPC facilitates this by printing out the address on the host during the recovery process. In some cases, this could be the only hint you would need to identify what is going wrong.

If you are running QNX, after an MMU fault occurs, run the 'sloginfo' utility in QNX to observe the output from the IPC resource manager:

```
***************** IPU-MMU Fault *****************
Jan 01 00:02:57    4    42     0 ****    addr: 0x96000000
Jan 01 00:02:57    4    42     0 ****    TRANSLATIONFAULT
Jan 01 00:02:57    4    42     0 ************************************************
Jan 01 00:02:57    4    42     0 syslink_error_cb: Received Error Callback for IPU1 : MMU Fault
Jan 01 00:02:57    4    42     0 syslink_error_cb: Scheduling recovery...
```

The address shown in the trace is the invalid virtual address accessed by the slave core.

On Linux, you should be able to obtain a similar message by running the 'dmesg' utility.

Halt the code when the fault happens

If the fault address is always the same, you can pre-map the location with some unused physical memory and setup CCS to halt upon access of the given address location. This is helpful so that you can take a look at the call stack, register values and memory contents. The procedure below illustrates what can be done on DSP1 of a DRA7xx device, but similar steps can be performed on other cores and other devices as well.

First, identify a 4KB aligned, unused area in the physical memory map that you can use. For sake of illustration, let's assume the 4KB starting at address 0xBA300000 is available and not used by any core in your program.

Before the fault happens -- possibly right after loading the slave core with IPC -- connect to the Debug DAP in CCS (Right-click in the Debug Window, and select 'Show all cores' to make it visible if it is hidden). The Debug DAP gives access to the physical memory as a whole. Bring up the Memory Window to inspect the MMU registers at 0x40D01000. Program up the MMU to map some unused memory to the fault address location by modifying the MMU registers to their corresponding values:

- MMU0 CAM (0x40D01058): 0x9600000E (Change the most significant 20 bits here to match your fault address. So it'd be 96000 if your fault address is 0x96000010 for example)
- MMU0 RAM (0x40D0105C): 0xBA300000 (Change the most significant 20 bits here to match an unused 4KB physical region in the memory map)
- MMU0 Lock (0x40D01050): 0x00000400
- MMU0 LD (0x40D01054): 0x00000001

If the fault happens on a data access

Connect to the DSP with CCS. Add a hardware watchpoint on the fault address:

- Right-click the Breakpoints window, select Hardware Watchpoint in the Breakpoint menu.
- Fill in the fault address location and choose memory access type. If you need both read and write accesses you can create a watchpoint for each type.

Let the program (both on the slave(s) and the host) run until the watchpoint is hit. You should have full access to the call stack, memory, and registers at that point.

If the fault happens on an instruction fetch

Make sure that the unused area specified by the MMU RAM register is zero-ed out (instruction of 0 is a NOP on the DSP). You can do this through the Debug DAP. E.g. for a fault address of 0x96000010, set physical location 0xBA300010 to 0.

Connect to the DSP with CCS. Add a breakpoint at the fault address.

Let the program (both on the slave(s) and the host) run until the breakpoint is hit when the MMU fault occurs. You should have full access to the memory and registers at that point.

"Service" the fault

If the fault is not always at the same location, it may be impossible to predict which memory location to pre-map and to set watchpoint/breakpoint on. In this case, you may want to let the fault happen, and then do some post-mortem analysis by connecting with CCS.

Disable recovery

In order to perform debugging after the MMU fault occurs, one needs to disable the automatic recovery mechanism, so that it does not wipe out the processor state immediately after the fault occurs.

On QNX

On IPC version 3.22 and above, you can simply invoke the 'ipc' command with the '-d' flag to disable recovery.

On older versions of IPC, you would need to modify the IPC source code. Comment out the lines in the function Ipc_recover() in <IPC_INSTALL_DIR>/qnx/src/ipc3x_dev/ti/syslink/build/Qnx/resmgr/syslink_main.c

<syntaxhighlight lang='c'> static void ipc_recover(Ptr args) { /*

```
  syslink_dev_t * dev = (syslink_dev_t *)args;
```

```
  deinit_ipc(dev, TRUE);
  init_ipc(dev, syslink_firmware, TRUE);
  deinit_syslink_trace_device(dev);
  init_syslink_trace_device(dev);
```

- /

} </syntaxhighlight>

Then rebuild IPC. Use this rebuilt version of the IPC resource manager for debugging purposes.

On Linux

There is a way to disable the recovery mechanism by simply writing to the debug filesystem:

```
% echo disabled > /debug/remoteproc/remoteproc<n>/recovery
```

where <n> is the core's remoteproc instance number. Echoing "enabled" will re-enable recovery.

If the fault happens on an instruction fetch

In this case, you can let the fault happen, map some memory to the fault location, set a software breakpoint at the fault address, clear the MMU fault, and let the slave halt on the breakpoint.

Let's again use DSP1 of a DRA7xx device as an example.

First, let the DSP run until it triggers the fault.

Connect to the Debug DAP in CCS (Right-click in the Debug Window, and select Show all cores to make it visible). Bring up the Memory Window to inspect the MMU registers at 0x40D01000.

Take a look at the MMU0 MMU_FAULT_AD register (0x40D01048). It contains the fault location -- its virtual address on the DSP. Write it down.

Modify the following registers to their corresponding values

- MMU0 CAM (0x40D01058): 0x9600000E (Change the most significant 20 bits here to match your fault address. So it'd be 96000 if your fault addresss is 0x96000010)
- MMU0 RAM (0x40D0105C): 0xBA300000 (Change the most significant 20 bits here to match an unused 4Kb physical region in the memory map)
- MMU0 Lock (0x40D01050): 0x00000400
- MMU0 LD (0x40D01054): 0x00000001

If the fault is due to an instruction fetch, write the breakpoint opcode '0x1001E000' to the location in the unused memory that was newly mapped in that corresponds to the fault address. e.g. in our example, for a fault address of 0x96000010, write the value to 0xBA300010.

This opcode corresponds to a software breakpoint instruction on a C674x DSP, and will serve to halt the program as soon as we clear the MMU fault. If you are on a core other than the DSP, you'll need to look up and use the opcode for the breakpoint instruction corresponding to its architecture.

Finally, clear the MMU fault interrupt by writing to this register:

- MMU0 IRQSTATUS (0x40D01018): 0x00000002 (Make sure you write to it, even if it shows the same value)

Writing to the MMU0 IRQSTATUS register clears the MMU fault interrupt and prompts the slave to continue on with execution.

Connect to the DSP in CCS and load the symbols for its executable. You should see the execution halted at the software breakpoint, and have full access to the memory and register contents. However, you will not have a call stack, since the fault location is not valid program memory based on the original program.

If the fault happens on a data access

In this case, you are somewhat out of luck. You can still follow the procedure for an invalid instruction fetch. However, instead of writing the breakpoint opcode to the fault location, you can try different values to see how the program behaves subsequently (0 would be an interesting value to try). Maybe it'd shed some light into which part of the code is making this illegal data access.

On some architectures, the MMU register MMU_FAULT_PC may contain the value of the program counter when the fault occurred. If so, you can simply put the breakpoint opcode for that architecture at that location before clearing the MMU fault. Unfortunately on the DRA7XX DSP this functionality does not seem to be available.