IPC Slave Error Recovery

Contents

Overview

The remote processors code, if not written properly, would cause a variety of exceptions or crashes like any other software. While the remote processor tracing can give diagnostic traces on the general code flow, it may not be enough to provide accurate information about crashes. Further, any active use-cases would experience a disruption in their execution. They should be able to gracefully errored out, and be able to reuse the remote processors. The remote core loader (remoteproc on Linux, ipc Resource Manager on QNX) provides the general infrastructure for different exception types and the ability to dump out useful information for debugging. On Linux, the recovery process is triggered by the remoteproc core, where the rpmsg virtio devices are destroyed and recreated.

The crashes on remote processors can be classified into three main types:

- MMU Faults - A remote processor MMU is unable to fetch an instruction or data address requested by the processor
- WatchDog Errors - The code on remote processor is stuck in a loop, and unable to perform any scheduling of other tasks
- Internal Exceptions - There can be some internal exceptions that are not exported outside of the remote processor subsystem

The following sections describe how the error notifications work along with the dump of the necessary exception information on DRA7XX devices.

MMU Faults

The remote processor MMUs have 32 TLBs, and can cover 4GB of address space. Hardware-assisted Table Walking Logic is also supported through L1 and L2 Page Table Entry (PTE) Tables. The L1 Table needs to be physically contiguous. These MMUs can generate an interrupt to the host processor on a variety of faults including a TLB Miss (useful only when Table Walking Logic is not enabled), or a Translation Fault (PTE not found).

On Linux, the remoteproc implementation manages the programming of these MMUs using the kernel's IOMMU framework. The generic remoteproc module programming of the IOMMU is done directly through the IOMMU domains. On QNX, this programming is automatically handled by the IPC resource manager based on the contents in the resource table.

The crash information is generated by the remote processors themselves. This generation is triggered slightly differently in the Cortex-M4 core and the DSP. For the DSP, there are special registers in the hardware to aid with MMU fault debugging. For M4, an internal bus error response is sent upon an MMU Fault and this is possible only when the *MMU_GP_REG* register is programmed properly. This causes the MMU Fault to generate an internal exception to the M4 core. SYS/BIOS provides the necessary exception handler implementations and can dump the processor registers and other information such as the executing task handle, its stack pointer and stack size into a user-provided buffer. This exception buffer is published to the remoteproc (or IPC resource manager for QNX) through the Resource Table.

Please look through the *Memory Management Units* chapter in device TRMs for more details on the above registers.

Watchdog

The remote processors are running SYS/BIOS, a RTOS that provides a simple scheduler based on hierarchical priorities. It is possible that a particular task may be running a busy loop and not yield the processor to execute other tasks. A Watchdog timer is used to detect this. There are no dedicated hardware watchdog timers for Cortex M4 cores, so two General Purpose Timers are used for detecting watchdog on each of the M4 cores.

> **NOTE**
>
> The following General Purpose Timer are powered up on newer 3.14 based TI Linux kernels. GPT9 & GPT4 for IPU2, GPT7 & GPT8 for IPU1, and GPT10 for DSP1 on DRA7XX

The Watchdog timer period is refreshed continuously on the remote processors by plugging in hooks into the SYS/BIOS scheduler. SYS/BIOS supports hook functions when switching Tasks or beginning a Swi. The SYS/BIOS has an Idle Task which is the lowest priority task, and is run when there are no active tasks. Both these hook functions and the Idle task would refresh the watchdog timer, postponing the interrupt/watchdog event as long as the scheduler is actively running and switching tasks.

The interrupt executes a Watchdog error handler function registered by the generic remote core loader through the platform-specific watchdog initialization hook. The handler function leverages the same remote core loader infrastructure as the MMU fault to dump out the relevant crash information. The remote core loader implementation manages these timers and an interrupt is generated to the host processor when the corresponding Watchdog timer expires.

In addition to the notification to and handling by remote core loader, the Watchdog expiration interrupt is also handled by SYS/BIOS. This interrupt is vectored to the #Device Exception Handler(DEH) module's exception handler, which prints details of the exception to the SYS/BIOS trace buffer, as well as dumping register contents to the Deh.outbuf buffer which can be viewed in CCS with the ROV tool. This remote core handling shows more information than the remote core loader's handler.

This "double handling" would seem to be either a conflict or superfluous, but it can be useful. Either "handler" can be used on its own since they're completely independent. The remote core loader handler will, by default, ultimately reset the remote processor, thereby possibly preventing the SYS/BIOS handler from running to completion, or at all (we don't really know since it's a race), and even if it runs to completion, the trace buffer contents will be wiped out upon the subsequent reset/reload of the remote core. The remoteproc's recovery mechanism can be disabled, in which case the SYS/BIOS handler will run to completion and the resulting output can be examined.

On Linux, the trace output of the remote core with the expired watchdog timer is also viewable even *after* recovery since remoteproc makes a copy of the trace output before resetting/reloading the remote core. This copy is present in a DEBUG_FS file named "**trace<n>_last**" in the corresponding remoteproc instance's DEBUG_FS directory. On QNX, the trace output can be optionally dumped into a file using the '-c' option of the IPC resource manager.

Remote Processor Exceptions

All the exceptions on the remote processors do not trigger an event/interrupt on the host processor directly. There are a number of internal core exceptions that generate an interrupt only to the specific remote processor. The Cortex-M4 core has about 16 internal interrupt events including various exception events like Non-maskable interrupt (NMI), Bus Fault, Usage Fault (refer to *Exceptions* chapter in the Cortex-M4 TRM for further information (http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/BABBGBEC.html)). All memory accessed by the remote processors have to be defined in an Attribute-MMU (AMMU), and the AMMU also generates a XLATE_MMU_FAULT interrupt to the core. The SYS/BIOS exception handler implementations are hooked to these exception interrupts and dump out the processor registers and other information such as the executing task handle, its stack pointer and stack size into a user-provided buffer.

On Linux, these exceptions are notified to the host processor by sending a special mailbox message to the rpmsg messaging bus layer. The rpmsg passes on this notification to the remoteproc, which then proceeds to perform a crash dump and recovery. On QNX, the host processor is similarly notified by this special mailbox message, and the IPC resource manager then proceeds to perform recovery.

> **NOTE**
>
> The AMMU is only relevant to the IPUs on DRA7xx.

Error Recovery

On Linux, remoteproc is responsible for only the device management of the remote processors, and provides interfaces to init/deinit a core. rpmsg component is the first client user of remoteproc today, and is responsible for causing a remote processor to be started and stopped. The rpmsg layer then cleans up after itself and deletes any existing devices. This in turn calls the remove functions of the various drivers hanging on the rpmsg bus. Any userspace operations on these client rpmsg drivers are errored out with a specific error allowing the userspace applications to gracefully clean up after themselves. The rpmsg virtio devices are destroyed and recreated. The recreation of the rpmsg virtio devices will cause the remote processor to be rebooted and rpmsg client devices are republished from the firmware running on the remote processors. Any rpmsg client drivers are re-probed, allowing them to export the driver interface. Synchronization in the rpmsg and remoteproc layers restrict the userspace applications from using a remote processor while the recovery is in progress.

This RPMsg design utilizes the bus infrastructure and driver model in the Linux kernel for performing the error recovery, and keeps the design simple while avoiding race conditions between open applications and drivers.

On QNX, the IPC resource manager, by itself, accomplishes the same combined role as remoteproc and the rpmsg component. All internal modules in the resource manager are torn down and recreated. One big difference between QNX and Linux is that on QNX all remote processors (not just the one with the error) that have been loaded are rebooted as all state information is wiped. This is an approach that guarantees that any existing communication between slave cores are reset and restarted.

Host OS

Error recovery is a stock feature of the remoteproc framework and is enabled by default. The kernel, once enable with remoteproc, will automatically reload the remote core on which an MMU fault has been detected.

To detect and recover from Exceptions on the remote core, the remote core must be configured to use #Device Exception Handler (DEH) module. The DEH module also allows for WatchDog support but must be also be enabled in the Linux kernel. On QNX the Watchdog support is always active and will interact with the DEH module if the latter is configured in.

Linux Kernel Config for Watchdog support

To enable Watchdog support, you must enable the following config in the kernel.

```
CONFIG_OMAP_REMOTEPROC_WATCHDOG=y
```

> **NOTE**
>
> The newer 3.14 based TI Linux kernels support IPU1, IPU2 and DSP1. Previous kernels only supported Watchdog on IPU2 on DRA7XX devices.

Disabling Recovery on Linux

There may be instances when the remote core ends up in a loop between crashing and reloading. Error recovery can be controlled with remoteproc's DEBUG_FS support. Each instance of remoteproc gets an associated DEBUG_FS file named "**recovery**" that can be written with either "**enabled**" or "**disabled**" as follows:

```
target# echo "disabled" > /debug/remoteproc/remoteproc0/recovery
target# cat /debug/remoteproc/remoteproc0/recovery
```

```
disabled
```

The remoteproc DEBUG_FS also supports an additional "**recover**" option useful to perform a one-time recovery if the current state is "**disabled**".

Disabling Recovery 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.

Host Application Recovery

> **NOTE**
>
> Beginning with IPC v3.36 or greater on Linux.

The transport Rpmsg layer for MessageQ properly detects and handles underlying socket failures. A socket failure typically occurs due to a crashed and/or reloaded remote core. The transport will inform the MessageQ layer of the failure, allowing MessageQ APIs to return back to the application with a new `MessageQ_E_SHUTDOWN` error code. This will allow the user level application to properly clean up and reestablish an IPC communication with a reloaded remote core without having to exit the application or restart LAD.

Both `MessageQ_get()` and `MessageQ_put()` will return the new error code. When `MessageQ_put()` returns an error, the application still owns the message and thus needs to free (`MessageQ_free()`) the message and properly close the MessageQ (`MessageQ_close()`). When `MessageQ_get()` returns an error, the application simply needs to delete the MessageQ (`MessageQ_delete()`).

To reestablish a connection with a recovered and reloaded remote core, the user application can detach (`Ipc_detach()`) from the reloaded core and reestablish a connection (`Ipc_attach()`) to the core until successful (`IPC_S_SUCCESS`). In some cases `Ipc_attach()` may require multiple attempts for success as the remote core completes its reload and restart process.

A test example illustrating this functionality can be found in the IPC product. For the host core, you can find it in **IPC_INSTALL_DIR/linux/src/tests/fault.c** (http://git.ti.com/cgit/cgit.cgi/ipc/ipcdev.git/tree/linux/src/tests/fault.c). The corresponding remote core application can be found in **IPC_INSTALL_DIR/packages/ti/ipc/test/fault.c** (http://git.ti.com/cgit/cgit.cgi/ipc/ipcdev.git/tree/packages/ti/ipc/tests/fault.c).

Device Exception Handler

DEH is a slave side module that handles exceptions and adds the necessary Watchdog timer support that must be configured on the host OS. Exceptions that are unrecoverable need to be communicated to the host processor so that it can print debug information, do resource cleanup and ultimately reload a slave processor. The notification mechanism for sending events to the host processor has been consolidated in this module.

The DEH module is provided as part of the IPC 3.x distribution as it is typically used in conjunction with devices that leverage IPC and remote loading features.

> **NOTE**
>
> Beginning with IPC v3.36 or greater, DEH has Watchdog timer configuration capabilities. By default DEH will use GPTimer 4 & 9 on IPU2, GPTimer 7 & 8 on IPU1, and GPTimer 10 on DSP1 for DRAA7XXX devices. In previous versions of DEH in IPC, only IPU2 (with GPTimer 4 & GPTimer 9) on DRA7XX devices was supported

Configuration

Below is a simple example to enable DEH (and default Watchdog timer) on a device's remote cores.

To enable DEH on the slave core, add the following to the remote core's configuration (*.cfg) file to handle exceptions. <syntaxhighlight lang='javascript'> var Deh = xdc.useModule('ti.deh.Deh'); </syntaxhighlight>

To add Watchdog detection on the DSP remote core, add the following to its configuration. <syntaxhighlight lang='javascript'> var Idle = xdc.useModule('ti.sysbios.knl.Idle'); Idle.addCoreFunc('&ti_deh_Deh_idleBegin'); </syntaxhighlight>

To add Watchdog detection on the M4 remote core in SMP mode, add the following to its configuration. <syntaxhighlight lang='javascript'> var Idle = xdc.useModule('ti.sysbios.knl.Idle'); Idle.addCoreFunc('&ti_deh_Deh_idleBegin', 0); Idle.addCoreFunc('&ti_deh_Deh_idleBegin', 1); </syntaxhighlight>

Watchdog slave support. If IPC power management is configured, make sure to place DEH Idle configuration functions before power management Idle functions.

If a different Watchdog is needed or desired (instead of the default) for a particular remote core, the following configuration lines can be added to the remote's configuration file. In this example, GPTimer 7 & 8 are being configured as Watchdodg timers and mapped to IRQs 60 & 61. There is also some Crossbar configuration being included to ensure the timers interrupts are properly routed to the appropriate core.

<syntaxhighlight lang='javascript'> var WD = xdc.useModule('ti.deh.Watchdog'); WD.timerIds.length = 2; WD.timerSettings.length = 2; WD.timerIds[0] = "GPTimer7"; WD.timerSettings[0].intNum = 60; WD.timerSettings[0].eventId = -1; WD.timerIds[1] = "GPTimer8"; WD.timerSettings[1].intNum = 61; WD.timerSettings[1].eventId = -1;

var Xbar = xdc.useModule('ti.sysbios.family.shared.vayu.IntXbar'); Xbar.connectIRQMeta(60, 38); Xbar.connectIRQMeta(61, 39); </syntaxhighlight>

Remote Core Debugging

The following articles describe interesting debugging techniques for issues detected by this error recovery feature:

- IPC MMU Fault Debugging
- Exception Dump Decoding

NOTE

When an error is detected, typically you will want to disable error recovery to debug the issue. #Disabling Recovery on Linux or #Disabling Recovery on QNX

Known Issues

- Error recovery is supported for IPU slaves using the Early Boot feature in Android starting in 6AL.1.1. Details on Early Boot are here - Linux | QNX.

{{ 1. switchcategory:MultiCore= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **IPC Slave Error Recovery** here.	Keystone= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **IPC Slave Error Recovery** here.	C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article* ***IPC Slave Error Recovery*** *here.*	DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article* ***IPC Slave Error Recovery*** *here.*	MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article* ***IPC Slave Error Recovery*** *here.*	OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article* ***IPC Slave Error Recovery*** *here.*	OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article* ***IPC Slave Error Recovery*** *here.*	MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article* ***IPC Slave Error Recovery*** *here.*	*For technical su please post you questions at http://e2e.ti.cor Please post on comments abo article* ***IPC Slav Error Recover*** }}

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters

DLP & MEMS
High-Reliability
Interface
Logic
Power Management

Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity

Retrieved from "https://processors.wiki.ti.com/index.php?title=IPC_Slave_Error_Recovery&oldid=224839"

This page was last edited on 17 February 2017, at 13:46.