AM335x EDMA Driver's Guide

From Texas Instruments Wiki

AM335x EDMA Driver's Guide

Linux PSP

Contents

Introduction

The enhanced direct memory access (EDMA3) controller's primary purpose is to service programmed data transfers between two memory-mapped slave endpoints on the device.

The EDMA3 controller consists of two principal blocks:

- EDMA3 channel controller (EDMA3CC)
- EDMA3 transfer controller(s) (EDMA3TC)

The EDMA3 channel controller serves as the user interface for the EDMA3 controller. The EDMA3CC includes parameter RAM (PaRAM), channel control registers, and interrupt control registers. The EDMA3CC serves to prioritize incoming software requests or events from peripherals, and submits transfer requests (TR) to the transfer controller.

The EDMA3 transfer controllers are slaves to the EDMA3 channel controller responsible for data movement. The transfer controller issues read/write commands to the source and destination addresses programmed for a given transfer. The operation is transparent to you.

Driver Configuration

- EDMA3 could be disabled/enabled from the following location during menuconfig.

start Linux Kernel Configuration tool.

```
make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm menuconfig
```

Building into Kernel

- Follow the *System Type* from the main menu. Then select *TI OMAP2/3/4 Specific Features*. Enable *OMAP3 EDMA*

support

```
System Type  --->
       TI OMAP2/3/4 Specific Features  --->
          <*>   OMAP3 EDMA support
```

CAUTION
As many modules use EDMA for data transfer, do not disable this option. Disabling this will break the compilation.

How to reserve different EDMA3 resources (DMA channels, TCCs, PaRAM Sets) on ARM side (region 0)

The resource allocation in EDMA3 is based on a pre-aligned static configuration. It is not possible for Linux running on Host ARM to restrict what resources are allocated by other masters. Each OS on every processor should restrict what resources they allocate to their clients based on the pre-aligned static configuration. Linux EDMA driver reserves resources to other masters so that they are not allocated to any client driver that requests them on ARM. The procedure to do so is explained below.

DMA Channels

In the source file **arch/arm/mach-omap2/devices.c**, array am33xx_dma_rsv_chans[][2] is used to reserve various DMA channels to the shadow regions of all other masters other than ARM. The first dimension is to mention the channel which is going to be reserved and the second dimension is to mention how many channels from that channel number will be reserved. So all other channels that are not reserved in this section can be used in the ARM side. See the code snippet below:

```
static const s16 am33xx_dma_rsv_chans[][2] = {
       /* (offset, number) */
       {0, 2},
       {14, 2},
       {26, 6},
       {48, 4},
       {56, 8},
       {-1, -1}
};
```

The first entry means 2 channels from channel '0' i.e. channels '0-1' will be reserved for usage of other masters.

To reserve/un-reserve the DMA channels, one has to modify this array accordingly, such that the reserved DMA channel(s) should NOT be used in the present core. All masters should work upon ONLY their subset of resources. In case of a conflict (different masters working on same set of resources), the system behavior is unpredictable.

TCCs (i.e Interrupt channels)

It is assumed that if a particular channel number x is reserved for a particular master then the TCC x is also reserved for that same master. All drivers and applications running on all masters must keep this in mind and be programmed.

PaRAM Sets

In the source file **arch/arm/mach-omap2/devices.c**, array am33xx_dma_rsv_slots[][2] is used to reserve various PARAM sets (slot) to the shadow regions of all other masters other than ARM. The first dimension is to mention the slot which is going to be reserved and the second dimension is to mention how many slots from that slot number will be reserved. So all other slots that are not reserved in this section can be used in the ARM side. See the code snippet below:

```
static const s16 am33xx_dma_rsv_slots[][2] = {
       /* (offset, number) */
       {0, 2},
       {14, 2},
```

```
        {26, 6},
        {48, 4},
        {56, 8},
        {64, 127},
        {-1, -1}
};
```

The first entry means 2 slots from slot '0' i.e. slots '0-1' will be reserved for usage of other masters.

To reserve/un-reserve the PARAM sets, one has to modify this array accordingly, such that the reserved slot(s) should NOT be used in the present core. All masters should work upon ONLY their subset of resources. In case of a conflict (different masters working on same set of resources), the system behavior is unpredictable.

Note 1: DMA channels and PaRAM Sets are one-to-one mapped, meaning thereby DMA channel X can ONLY use PaRAM Set X for its working. The system integrator should make sure that the DMA channels reserved to ARM side should also have the corresponding PaRAM Sets (at least) reserved to ARM side. Other PaRAM Sets available on the ARM side (lying between 64-255) could be used for link purpose. So they can be independently reserved/unreserved to ARM side.

Note 2: A DMA/QDMA channel also require a TCC for correct functioning. By default, DMA channels and TCCs are one-to-one mapped, meaning thereby DMA channel Y can ONLY use TCC Y for its working. (No such constraint is there for QDMA channels.) The system integrator should assume that the DMA channels reserved to ARM side, also have the corresponding TCCs (at least) reserved to ARM side. In case user passes a specific TCC to be used by the DMA/QDMA channel, he/she should first check its availability on ARM side.

How to change various other global settings

How to change default event queue (or Transfer Controller, TC) priorities?

Array am33xx_queue_priority_mapping[][2] can be used to change the default event queue priorities. This array is an array of structures having the first structure member as the event queue number and the second structure member as the priority. '0' is the highest priority, '1' is the second highest and so on. See the code snippet below:

```
static const s8 am33xx_queue_priority_mapping[][2] = {
        /* {event queue no, Priority} */
        {0, 0},
        {1, 1},
        {2, 2},
        {3, 3},
        {-1, -1}
};
```

The system integrator can appropriately modify this array, keeping bandwidth limitations (for different users and whole system in general) in mind.

Present configuration of EDMA3 resources on AM335x platform

EDMA3 Resource	**AM335x**	
	ARM	**Other Masters**
DMA Channel	2-13, 16-24, 36-47, 52-55	0-1, 14-13, 26-31, 48-51, 56-63
QDMA Channel	0,1	2-7
TCC	2-13, 16-24, 36-47, 52-55	0-1, 14-13, 26-31, 48-51, 56-63
PaRAM Set	2-13, 16-24, 36-47, 52-55, 64-127	0-1, 14-13, 26-31, 48-51, 56-63 , 128 - 255

How to Program the Cross bar events

AM335x has **95 EDMA trigger events** while it only has **64 DMA channels**. The first 64 events are by default routed to trigger the 64 DMA channels (One to one mapping, i.e. event 10 is routed to channel 10). The rest 31 events (called the

cross bar events) are routed to any of the 64 channels by a cross bar switch. **Programming of the cross bar is available at the system control module level**.

EDMA driver takes care of this routing via the array am33xx_xbar_event_mapping[]. When a request for a channel is made for an event of event number > 63, then the channel allocation is based on the below mentioned array. The first dimension is the cross bar event number which is essentially the actual event number - 63. The second dimension is the channel number to which you want the event to be routed. See the code snippet below:

```
static struct event_to_channel_map am33xx_xbar_event_mapping[] = {
        /* {xbar event no, Channel} */
        {1, -1},
        {2, -1},
        {3, -1},
        {4, -1},
        {5, -1},
        {6, -1},
        {7, -1},
        {8, -1},
        {9, -1},
        {10, -1},
        {11, -1},
        {12, -1},
        {13, -1},
        {14, -1},
        {15, -1},
        {16, -1},
        {17, -1},
        {18, -1},
        {19, -1},
        {20, -1},
        {21, -1},
        {22, -1},
        {23, -1},
        {24, -1},
        {25, -1},
        {26, -1},
        {27, -1},
        {28, -1},
        {29, -1},
        {30, -1},
        {31, -1},
        {-1, -1}
};
```

To route the event 81 through channel 16, one has to modify the above array as {18,16}, instead of {18,-1}*.(81-63=18)*

The system integrator has to modify this array, keeping in mind that the second dimension should be an integer between 0-63. He should also make sure that if channel x is used for cross bar events then its default event, event x is not required.

Sample test application for EDMA Driver

EDMA Driver APIs are available for use only in the kernel space. Hence the sample application provided must be built as a kernel module and used.

The sample EDMA application is available along with other Module examples as a tar archive in *src* directory of PSP release package (Directory Structure of PSP release package can be seen here AM335x PSP Package Contents)

Copy this sample application file in a location (preferably not inside the kernel source code) and build it as a kernel module using a make file similar to one shown below

```
obj-m = edma_test.o
KDIR = "../linux-kernel"
all:
        make -C $(KDIR) M=$(PWD) ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- modules

clean:
        make -C $(KDIR) M=$(PWD) clean
```

Insert this kernel module once the kernel is up, to see the sample application working.

Links

Amplifiers & Linear (http://www.ti.com/lsds/ti/analog/amplifier_and_linear.page)
Audio (http://www.ti.com/lsds/ti/analog/audio/audio_overview.page)
Broadband RF/IF & Digital Radio (http://www.ti.com/lsds/ti/analog/rfif.page)
Clocks & Timers (http://www.ti.com/lsds/ti/analog/clocksandtimers/clocks_and_timers.page)
Data Converters (http://www.ti.com/lsds/ti/analog/dataconverters/data_converter.page)

DLP & MEMS (http://www.ti.com/lsds/ti/analog/mems/mems.page)
High-Reliability (http://www.ti.com/lsds/ti/analog/high_reliability.page)
Interface (http://www.ti.com/lsds/ti/analog/interface/interface.page)
Logic (http://www.ti.com/lsds/ti/logic/home_overview.page)
Power Management (http://www.ti.com/lsds/ti/analog/powermanagement/power_portal.page)

Processors (http://www.ti.com/lsds/ti/dsp/embedded_processor.page)

- ARM Processors (http://www.ti.com/lsds/ti/dsp/arm.page)
- Digital Signal Processors (DSP) (http://www.ti.com/lsds/ti/dsp/home.page)
- Microcontrollers (MCU) (http://www.ti.com/lsds/ti/microcontroller/home.page)
- OMAP Applications Processors (http://www.ti.com/lsds/ti/omap-applications-processors/the-omap-experience.page)

Switches & Multiplexers (http://www.ti.com/lsds/ti/analog/switches_and_multiplexers.page)
Temperature Sensors & Control ICs (http://www.ti.com/lsds/ti/analog/temperature_sensor.page)
Wireless Connectivity (http://focus.ti.com/wireless/docs/wirelessoverview.tsp?familyId=2003&sectionId=646&tabId=2735)

Retrieved from "http://processors.wiki.ti.com/index.php?title=AM335x_EDMA_Driver%27s_Guide&oldid=169760"

Categories: PSP | AM335x | EDMA3 | Linux