CONFIDENTIAL

Description of DICE JR/Mini Mixer bug and potential workaround

File: tcd22x0MixerBug.doc
Printed: 2/17/2008 8:04:00 PM
Updated: 2/15/2008 7:23:00 PM

Revision history

When	Who	What
2008-02-13	ML	Document Created
2008-02-17	ML	Added implementation specific information

Table of Contents

1 Introduction

The bug in the Mixer only comes into play when the MIXER_NUMOFCH is different from 16. In general there is no reason to set it to anything else. Unfortunately this register is forced to 8 when operating in high mode. The result is that using the usual routing of data only every second sample will be calculated and every other will be what ever is in the ram from the last time the mixer operated at 16 channels (before changing to high rate).

The symptoms will be:

- A DC might be present if the mixer was not cleared for at least two sample periods before changing to high mode.
- The signal level will be down by 6dB due to every second sample missing.
- High frequency aliasing can result from every second sample missing.

This is not an acceptable scenario and there are a number of workarounds which will be discussed later in this document.

2 Details of operation

The mixer is working based on a ping/pong buffer scheme. The bugger halves are swapped at every negative edge of 1fs. One buffer is available to the mixer to write results into while the other is ready for the router to read from.

All devices in the DICE router system changes buffer at 1fs neg edge so the router can work independently of the devices and have all clock cycles within a 1fs frame (except e few guard cycles in the ends) available for processing.

2.1 The mixer problem

The two 16 channel ping/pong buffers are implemented in one 32 channel RAM. A buffer variable which is either 0 or 1 will determine which part the router is accessing and which part the mixer is accessing.

The router will always access either the first 16 or the last 16 addresses. So will the mixer when in 16 channel mode but unfortunately when in 8 channel mode it will change between using the first 8 or the last 8. That is of course not good considering the router always assuming 16 channels boundary for the two buffers.

That is the reason why every second sample is missing as the upper half of the ping/pong buffer is never written and the router uses that every second sample.

The good thing; however is that the mixer does store those missing samples, just in the wrong place.

Looking at the sequence of events in this case:

1) buffer=0, right after 1fs negedge
2) Mixer puts results into addr[0..7] of first buffer
3) At the same time router reads the other buffer, addr[16..31] which is never written, it actually only reads the channels routed.
4) Mixer done, router done at some point before next frame
5) Buffer=1, right after 1fs negedge
6) Mixer puts results into addr[8..15], still first buffer.

7) At the same time router reads the first buffer, addr[0..15] which has the correct sample values in addr[0..7] while the mixer is writing into the same buffer at addr[8..15]. Those upper channels are the ones we should have gotten in the next pass instead.
8) Mixer done, router done at some point before next frame
9) Go back to 1)

So the data we are missing every second sample are actually available in ch8-15 (router point of view) in the previous cycle if we make sure to access them late in the router.

2.2 Workaround with 8 ch of mixed data (not perfect)

This workaround will create all 8 channels of mixed data but it will be based on copying neighboring samples two and two. The effect of this will be:

- Energy above 48k will alias (probably not a big problem)
- The RMS of the in-band signal will be correct
- A small amount of energy might be created between 48k and 96k, (probably not a big problem).

The principle for this workaround is to route mixer outputs to the APB interface. A very simple FIQ routine will copy the data back accordingly.

One challenge is to detect which of every second sample is the correct one after changing clocks etc. This is what we know:

- The samples we should skip are constant (RAM never written)
- The samples we want to use are changing or constant

The solution is the following:

- At every interrupt we receive 4 samples of 8 channels of data
- Of those samples are good (0,2 or 1,3) and two are bad.
- We hold a global variable telling if we should use (0,2 or 1,3).
- At every interrupt we check if the ones we consider bad are non zero.
- If they are we flip the global variable.

This means that after a rate change etc. we will get the data correct at the first non zero sample (we will make sure the bad data are 0).

What the FIQ does on top of this:

- Either copy 0 to 1 and 2 to 3 or copy 1 to 0 and 3 to 2.

2.3 Workaround with 4 ch of mixed data (perfect)

This workaround will create 4 channels of mixed data. Those 4 mixes will be the correct 192k audio values. The only penalty is adding a few samples of latency.

The principle for this workaround is to route mixer outputs to the APB interface. A very simple FIQ routine will copy the data back accordingly.

The routing to the APB interface must follow the following rules:

- Route Mixer Ch0-3 to APB Ch0-3
- Route Mixer Ch8-11 to APB Ch4-7
- The last Routes must be placed in the router after entry 20 (file up with mute entries if needed)
- Route APB Ch0-3 to wherever the mixes should go

This principle requires the same detection of good and bad samples as described in the previous workaround so that is not repeated here.

What the FIQ does on top of that is either:

- Copy sample0 Ch0-3 to sample0 Ch0-3
- Copy sample0 Ch4-7 to sample1 Ch0-3
- Copy sample2 Ch0-3 to sample2 Ch0-3
- Copy sample2 Ch4-7 to sample3 Ch0-3

Or:

- Copy sample1 Ch0-3 to sample0 Ch0-3
- Copy sample1 Ch4-7 to sample1 Ch0-3
- Copy sample3 Ch0-3 to sample2 Ch0-3
- Copy sample3 Ch4-7 to sample3 Ch0-3

2.3.1 Placing the APB route late in the router

There are basically three ways of setting the router in the DICE products:

- Using the **dal** functions
- Using a proprietary scheme writing directly to the router
- Through EAP (Extended Application Protocol)

When using the **dal** router functions the entry will always be placed late in the router as long as the Mixer and at least one other device are part of the domain list of output devices. This will always be the case, so this will not need to be considered. The firmware part setting up the routing must route the mixer outputs to the APB interface and then route the output of the APB to where the

When using a proprietary scheme make sure that the routing of the mixer channels 4-7 are placed after entry 20 in the router. If the mixer is placed first, that will take up the first 18 places, so one other device before the APB routes will fix the problem.

When using EAP an automatic method is available by defining some flags. This will be described later.

3 Implementation

The implementation resides in the DICE and DAL chip specific directories.

Two new files have been added to the DICE group – `mixfix.c` and `mixfix.h`.

Spoofing code has been added to `dalRouter.c` so existing firmware would not need to change as the router will automatically route the outputs of the mixer through the ARM APB interface.

The following defines controls the workaround code:

_FIQ:	Must be defined for any level of this fix. This define will make the FIQ handler available in the interrupt module.
_MIXFIX_FIQ:	Includes the mixfix FIQ handler code and will initialize and install the handler. The FIQ will be disabled until manually enabled with a call to `mixfixEnable`. This define alone will not touch the router.
_MIXFIX_PERFECT4:	This flag affects all levels of this implementation and determines which of the workarounds described above will be installed.
_MIXFIX_DAL:	This flag makes the workaround automatic for applications relying on dalRouter for routing. The code will automatically enable the FIQ when in high mode if the mixer is a member of the domain and it will spoof all routing from the mixer and it will automatically add routes from the mixer to the ARM APB interface.
_MIXFIX_EAP:	This flag makes the workaround automatic for applications relying on EAP for routing. EAP is not yet added to the SVN source tree, but it will be in the near future.

For a typical DAL based application add the following line to `Make.params`:

```
CFLAGS += -D_MIXFIX_FIQ -D_MIXFIX_DAL -D_MIXFIX_PERFECT4 -D_FIQ
```

NOTE: For system using a proprietary build structure please make sure that the `sram` memory section is defined in the .ld file used. This will assure that the code is placed in internal sram which is essential for performance.

