9/13/00

# Errata: EP7212 Rev D

EP7212 High-Performance, Low Power System-on-Chip with LCD Controller and DAI
(DS474PP1, FEB '00)

1) DRAM refresh

**Description:** Under some circumstances, the bus can become saturated which will result in stalling the DRAM refresh signals. This causes data in the DRAM to become invalid. In order to get a refresh cycle generated correctly the memory controller state machine has to be in the idle state for one cycle after the end of the extended RAS precharge period with no DRAM or ROM requests active.

**Workarounds:**

A) Set the clock to 36 MHz. No observed refresh faults occur at 36 MHz and below.

B) Avoid using DRAM for *both* program and data storage. If code is run out of SRAM or ROM, this will tend to keep the DRAM activity from saturating the bus.

C) Make certain that cache is enabled for DRAM.

D) Allow some time to free the bus from constant DRAM activity so that a refresh can occur. If the above guidelines still cause DRAM read and write errors, then set up an fast interrupt that runs in internal SRAM. For example, the DAI interrupt service routine that is used by the MP3 player routine is called at an 11 kHz rate. If this routine is placed into internal SRAM, then on each interrupt, the external bus becomes idle long enough to allow the refresh logic to function properly.

E) Add a software refresh routine. This routine can be called during each OS timer tick (typically every 10 msec). This is probably the best option if: 1) clock speed is >36 MHz, **and** 2) program is running out of DRAM. The following code is one possible means for generating a software refresh.

```
#define DRAM_START    0x00000000
#define DRAM_END      0x01000000
#define DRAM_ROW_SIZE 0x00000400
#define DRAM_REFRESH (((DRAM_END-DRAM_START)/DRAM_ROW_SIZE)+99)/100
static void
do_DRAM_refresh(void)
{
    static int *row_ptr;
    volatile int val;
    int i;
    for (i = 0;  i < DRAM_REFRESH;  i++) {
        val = *row_ptr;
        row_ptr += DRAM_ROW_SIZE / sizeof(*row_ptr);
       if (row_ptr >= (int *)DRAM_END) row_ptr = (int*)DRAM_START;
       }
}
```

**Cirrus Logic, Inc.**
P.O. Box 17847, Austin, Texas 78760
(512) 445 7222 FAX: (512) 445 7581
http://www.cirrus.com

2) Thumb mode operation

**Description:** Thumb mode will not function correctly when running code out of DRAM and if the cache is disabled. The error occurs only with cache disabled. The error occurs when entering Thumb state on an odd address. The CAS and RAS signals are not asserted, and therefore the read of the instruction will only fetch the higher byte with the lower byte being ignored.

**Workaround:** Make certain cache is enabled. With cache enabled, only aligned accesses are issued.

3) LRCLK duty cycle

**Description:** LRCLK is not a 50% duty cycle clock, the actual duty cycle of LRCLK is 63/128% low and 65/128% high. Data is aligned with LRCLK.

**Workaround:** None, Data is aligned with LRCLK, many standard ADC and DACs (CS43L40, CS43L42, CS43L43, and CS53L32) align data with LRCLK. Verify that the CODEC used aligns with LRCLK and does not require a 50% duty cycle.

4) DAI input timing

**Description:** When receiving data via the DAI port, the device will capture data one clock prior to LRCLK falling.

**Workaround:**

A) Perform a single bit left shift on the Right Channel Data (when LRCLK is low). The result will be 15 bits of valid data.

B) If interfacing with a CS53L32 ADC, the CS53L32 presents a zero followed by valid data after the rising edge of LRCLK. Once the data has been captured, a single bit left shift for both Left and Right Channel Data will provide 15 valid bits of data from the ADC.

5) SSI1 port operation

**Description:**, The SSI1 port will transmit incorrect data if bits 4:7 form an even number and bits 0:3 are **not** equal to 0xF. For example 0x12 will cause an error, but the values 0x1F and 0x22 will not cause an error.

**Workaround:** The SSI1 port can be used for receiving data with no errors. If transmitting data, avoid the condition in the description.. Alternatively, you can use SSI1 in Extended mode.

6) nURESET operation

**Description:** If nURESET is asserted before the processor is operating after a wakeup from Standby mode, the system will enter into an unknown state. When this occurs, the only recovery is to assert the nPOR.

**Workarounds:**

A) Do not use the nURESET signal. Tie the nURESET to $V_{dd}$.

B) If nURESET must be used, make certain that it cannot be asserted until after the processor is running. It is necessary to gate uRESET with a run condition flag, which could be a GPIO pin that is asserted via software, or a flip-flop that is set from an external chip select such as nCSx. Under no circumstances should the RUN signal be used. The RUN signal goes high approximately 128 msec after WAKEUP is asserted. Instructions are not fetched from external ROM (nCS0) until approximately 150 msec to 350 msec after WAKEUP.

7) FASTWAKE operation:

**Description:** The FASTWAKE feature does not meet the performance described in the data sheet. When the FASTWAKE bit of SYSCON3 is set, the EP7209 runs at 256 Hz for approximately 10 instructions or approximately 39 msec after the processor speed is increased to 36 MHz.

**Workaround:** There is no workaround; therefore, the FASTWAKE feature is being removed from the data sheet.

For any questions regarding this Errata, please send email to: epdapps@crystal.cirrus.com