Engineering Design Services Case Study Report

## Buffer Size Management

To interface audio data stored on the SD card (organized using FATFS) to the Altera Audio IP core’s FIFO space (which then is spooled at an appropriate sampling rate to the *audio codec*), it is important to understand the organization of data stored on the SD card. Data stored on the SD card is stored in a sequential byte-by-byte fashion. Each *.WA*V file on the SD card, has audio samples packed as 16 bits. The first 16 bits is allocated to the left channel, the next 16 bits are allocated to the right channel and so on in this alternating pattern. The reason why a buffer is required is because the Altera Max10 operates at a significantly larger frequency than the SD. Thus, a buffer of type uint8\_t (8 bits/1 byte) will be implemented to extract and store all bytes of a given file in intervals of a chosen buffer size number of bytes from the SD card. The type, unit8\_t, is a data type corresponding to Altera that holds 8 bits. Since, the SD card (of FATFS) uses little endian ordering, the data fed into the left and right buffers must be reorganized into big endian to ensure proper functionality of the audio player.

The buffer size that was implemented was 1024 bytes. This buffer size allowed for the mitigation of the speed mismatch between the SD card and the FPGA. It is important to understand that directly transferring data between two clock domains of significantly different speeds can lead to *metastability*. Metastability occurs when transitions in clock signals occur in the sender during set up and hold times of the receiver and vice versa. In the case of this project, the sender (the SD card) operates at a much slower rate than the receiver (Audio IP core). If the clock domains do not virtually align, then there will be no synchronization. This will possibly result in metastability and “noise” to be spooled out on the audio codec. When one entity is trying to receive data the other could be idle and when one entity is sending data the other can be idle. When this occurs, there is no synchronization and therefore no *valid* data transmission. By implementing a buffer, it will prevent the FPGA from idling when it is waiting for new data.

### Buffer Size

The buffer size of 1024 was strategically chosen. Given that the original buffer size was a power of 2 value, various power of 2 values were experimented with. From analysis and experimentation, it was determined that