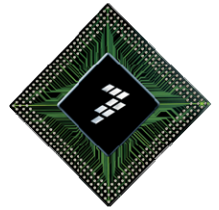
iMX51 Audio Driver for Windows CE 6.0

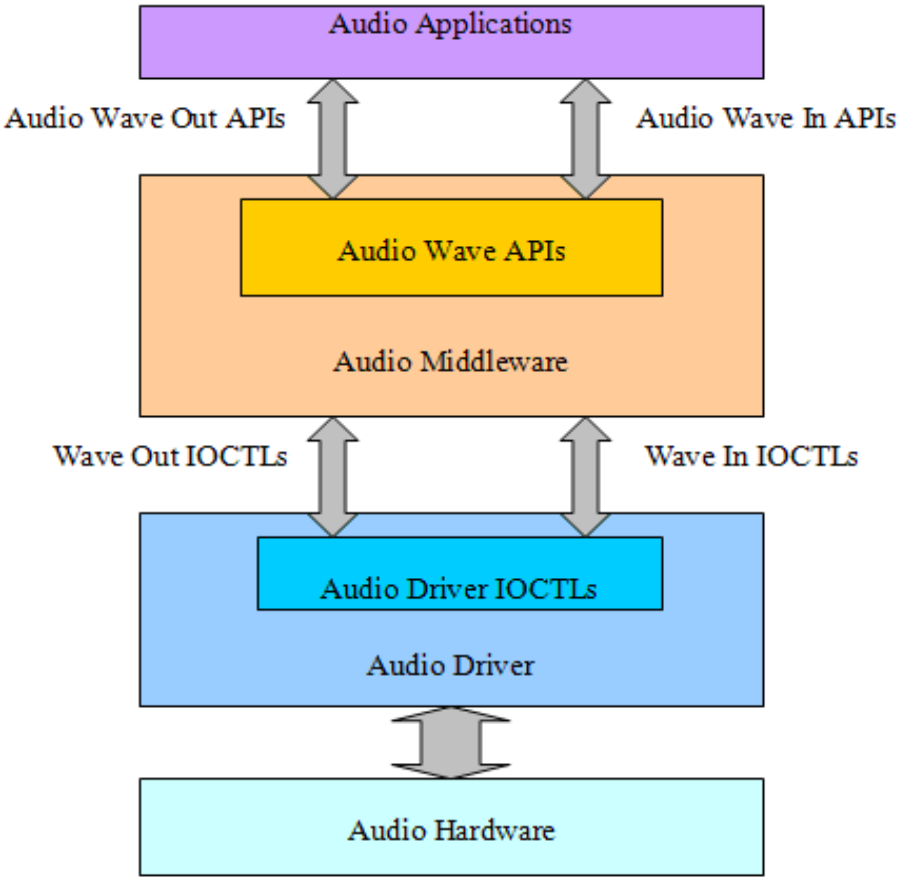


freescale™
semiconductor

Windows CE 6.0 Audio Stack Architecture

- For Windows CE 6.0, the audio stack enables functionality of an audio device allowing the operation system (OS) to support audio playback and audio record.

- The audio stack components can be divided into three levels: the audio applications, audio middleware and audio driver. Figure on the right shows the hierarchy of the audio stack.



freescale™ semiconductor

Audio Application

• The audio application is responsible for controlling the output/input audio stream flow. For Windows CE 6.0, standard audio interfaces called audio wave APIs are defined by Microsoft. Audio applications can call these interfaces to control the output/input audio stream. A typical audio application may contains following steps for audio stream playback/record:

- Audio application calls WaveOutOpen ()/WaveInOpen () function to open the output/input device
- Audio application allocates buffers for storing the output/input audio stream data and then calls waveInAddBuffer ()/waveOutWrite () to submit the buffer to audio middleware software for further processing.
- When the allocated output/input streams are processed by the low level software (middleware and audio driver), the low level software will set event to inform the audio application that the processing is finished. After receiving the event, the audio application can then submit another output/input buffers to middleware for processing.

Audio Middleware

• The audio middleware software functions as the bridge between the audio applications and the audio driver. For audio applications, the middleware software provides standard wave APIs. For audio driver, the middleware software my issue IOCTLs according to the wave APIs that are processing. For the typical audio application depicted above, the middleware may do following operations:

- When the audio application calls WaveOutOpen()/WaveInOpen() function to open the output/input device, the middleware software will issue IOCTL message WODM_OPEN/ WIDM_OPEN to the audio driver respectively

- When the audio application calls waveInAddBuffer()/waveOutWrite() to submit the buffer, the middleware software will issue IOCTL message WIDM_ADDBUFFER/ WODM_WRITE to the audio driver respectively

freescale™ semiconductor

Audio Driver --- WAVEDEV2 Module

• The audio driver controls the audio hardware directly and provides standard IOCTLs defined by Microsoft to the middleware. The audio driver also maintains buffers allocated by the audio applications.

• i.MX51 audio driver adopts WAVEDEV2 Module defined by Microsoft. For WAVEDEV2 module, the whole audio driver is implemented in the BSP, no MDD driver is included.

• The hardware platform related to the audio is abstracted as the **hardware context** by the WAVEDEV2 model. Each hardware context will contain two **device context**s: the **input device context** and the **output device context**. The input device context is for audio input channel and the output device context is for audio output channel.

• Every time when the audio application calls WaveOutOpen ()/WaveInOpen () function to open the output/input device, the **stream context** object will be created and linked to the corresponding device context. For each device context (input device context and output device context), there is a list to maintain the stream context. Following figure shows the stream context list for the device context.

Audio Driver --- WAVEDEV2 Module

- Common code for WAVEDEV2 driver:

PLATFORM\COMMON\SRC\SOC\COMMON_FSL_V2_PDK1_7\WAVEDEV2

- iMX51 CSP code, SSI and Audiomux address, DMA irq:

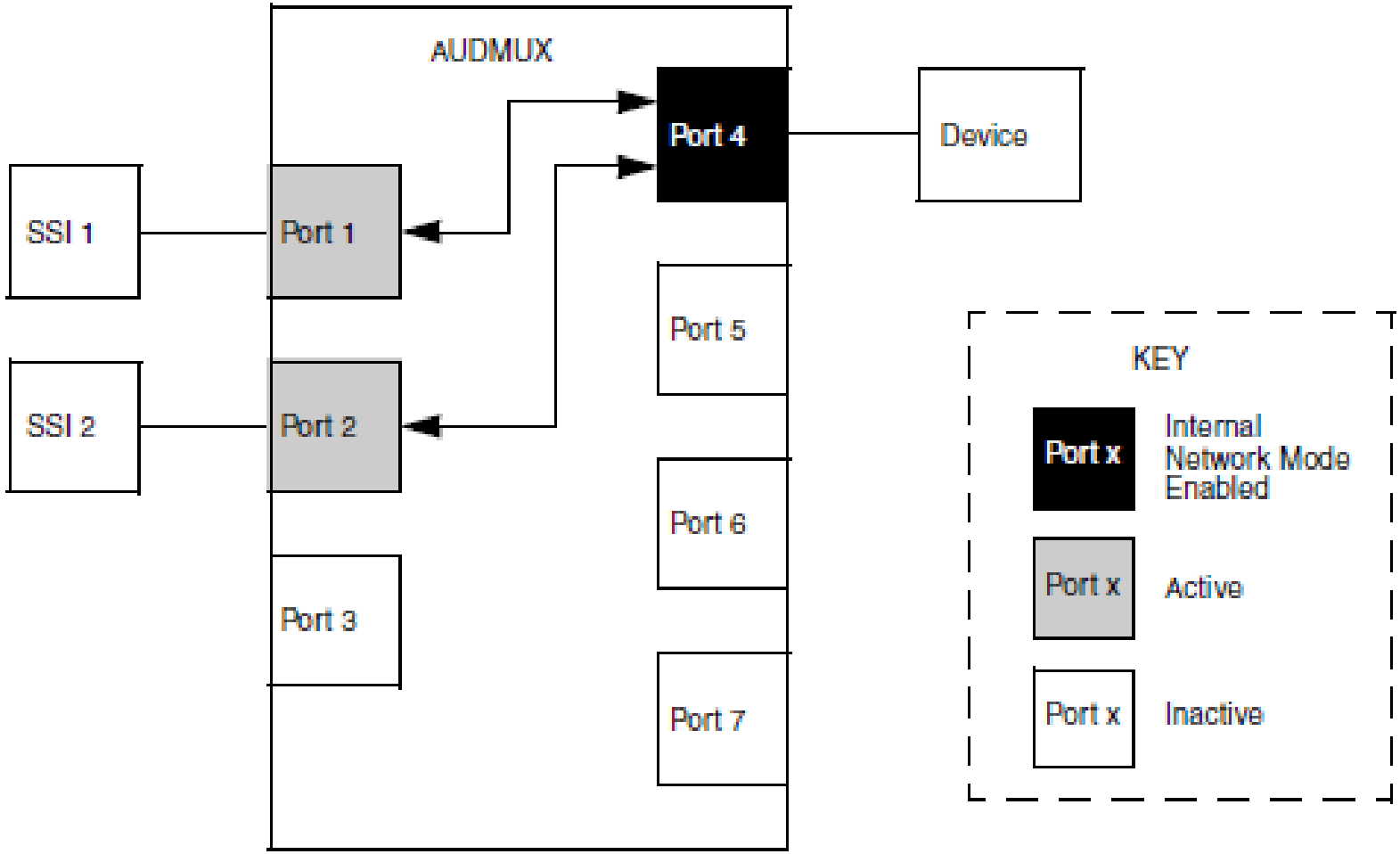
PLATFORM\COMMON\SRC\SOC\MX51_FSL_V2_PDK1_7\WAVEDEV2

- Hardware audio code related code:

PLATFORM\iMX51-EVK-PDK1_7\SRC\DRIVERS\WAVEDEV2\SGTL5000

- How to support a new audio codec

MX51 Audio MUX



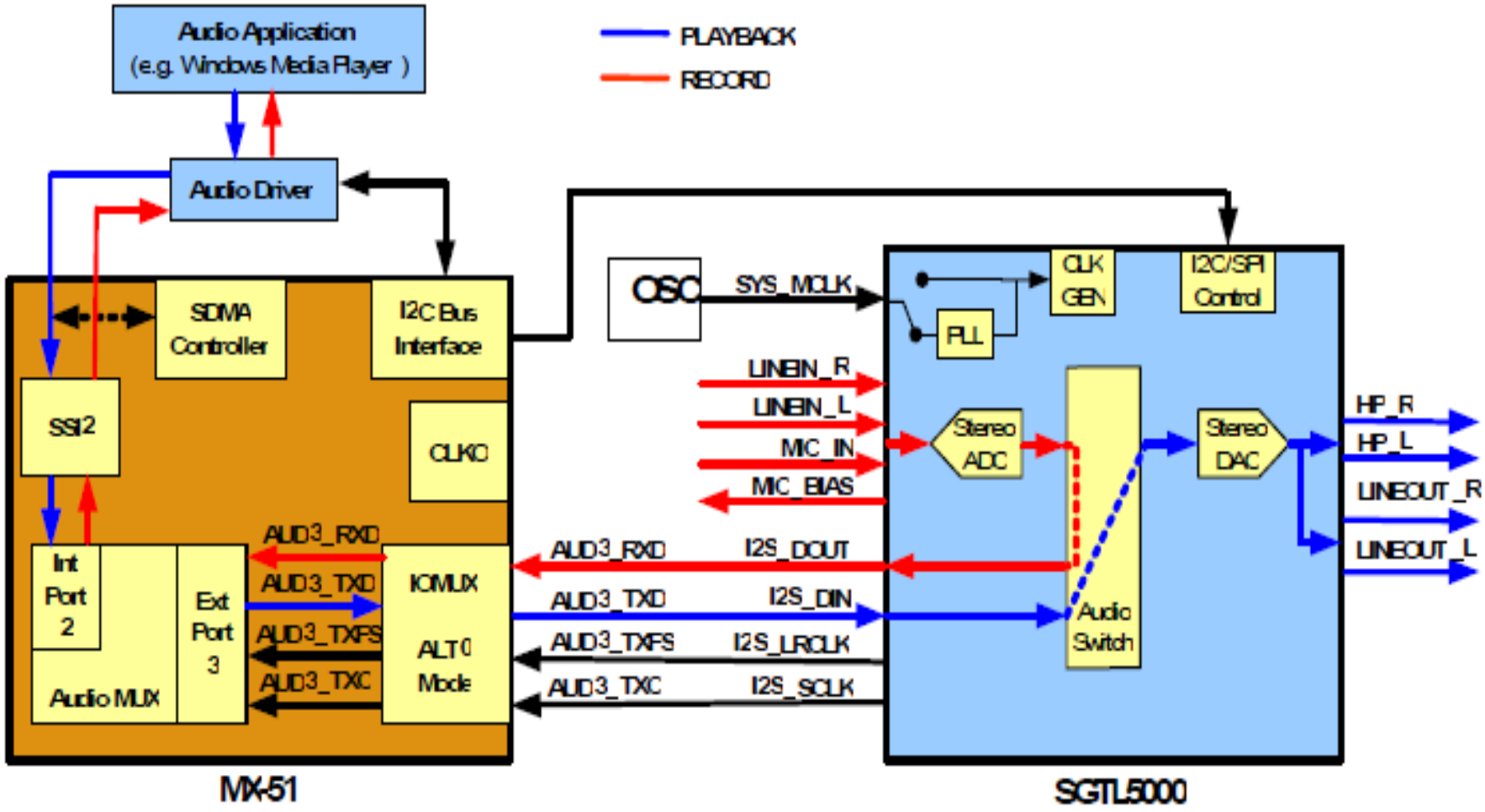
freescale™ semiconductor

MX51 Hardware Support for Audio

- MX51 provides 6 SSI ports, including 2 internal ports and 4 external ports. Totally 4 SSI hardware channels are available for hardware
- Each SSI channel can be configured to work in I2S mode or AC97 mode
- Audio Mux to support programmable interconnection between SSI internal ports and external ports
- SPDIF interface support (For MX51, only SPDIF TX is supported)
  - pwfx->wFormatTag = WAVE_FORMAT_WMASPDIF; // SPDIF FORMAT

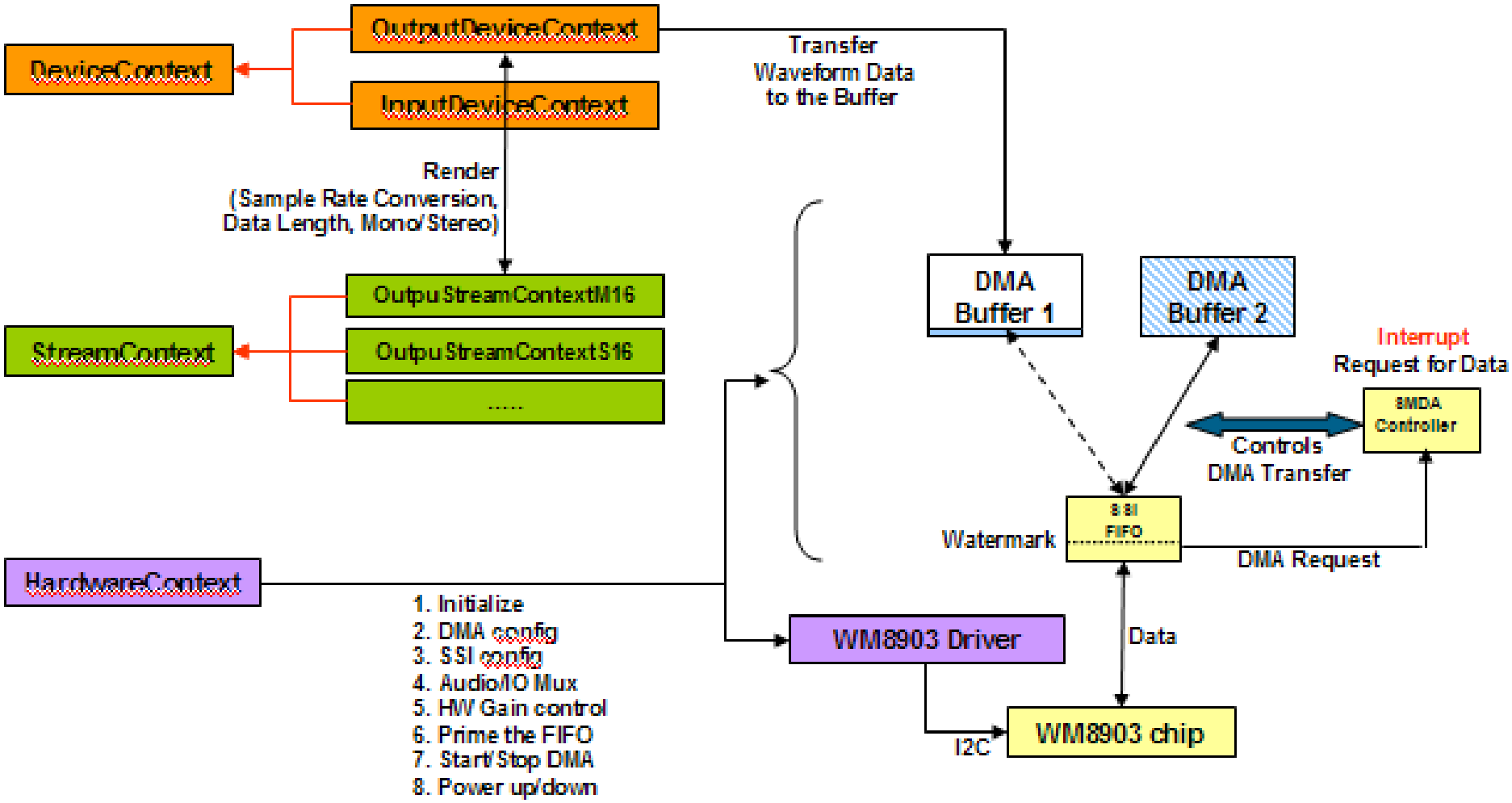
- Audio CODEC: SGTL5000
  - I2S audio data transfer, two channels
  - Hardware sample rate 44.1 K bps
  - Data width for each channel: 16 bits
  - I2C control, iMX51 I2C port 2 is reserved for codec control

MX51 Audio Hardware Connection



freescale™ semiconductor

MX51 Audio Driver for SGTL5000



freescale™ semiconductor

MX51 Audio Driver for SGTL5000 --- DMA Operation

- The audio driver sends data from system memory to SSI FIFO or receives data from SSI FIFO to system memory by DMA operations.

- The audio driver allocates four buffers from system memory to support the DMA operations, two buffers for audio recording and two buffers for audio playback.

- Why two buffers for each channel (output & input)

MX51 Audio Driver for SGTL5000 --- DMA Operation

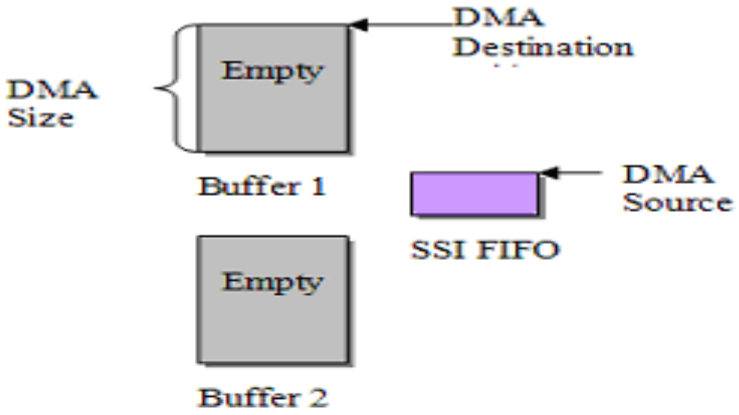


Figure a

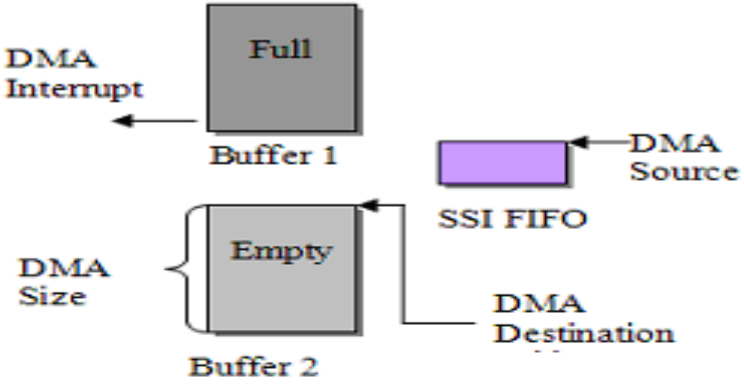


Figure b

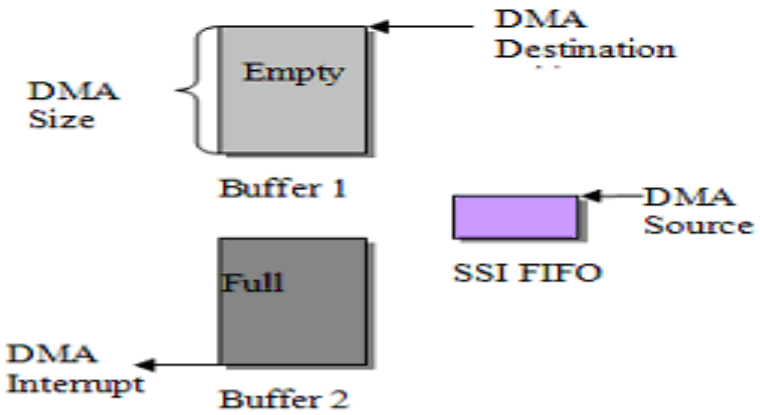


Figure c

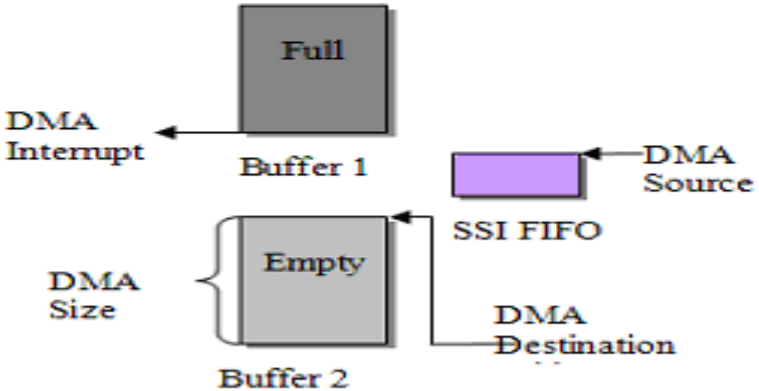


Figure d

freescale™ semiconductor

MX51 Audio Driver for SGTL5000 --- DMA Operation

• Marks the two DMA buffers for audio recording as buffer 1 and buffer 2. After initialization both buffer 1 and buffer 2 are empty, as shown by figure a. Then audio driver performs DMA operations by following steps to get audio recording data

• Sets DMA source address as the SSI FIFO start address, DMA destination address as the buffer 1 start address and DMA size as the size of buffer 1. Once the DMA parameters (source, destination address and transfer size) are configured, audio driver will invoke DDK function to start the DMA transfer. This part can be illustrated by Figure a.

• Once DMA buffer transfer for buffer 1 is started, audio driver will set the DMA destination address to the start address of buffer 2. After DMA transfer for buffer 1 is finished, the DMA interrupt will occur and the DMA interrupt handler inside audio driver gets chance to run. At the same time, DMA hardware will begin the DMA transfer for buffer 2 automatically. DMA interrupt handler for buffer 1 copies data from DMA buffer 1 to the buffers allocated by audio applications. The operation process for this step can be illustrated by Figure b.

• DMA interrupt handler for buffer 1 will set the DMA destination address to the start address of buffer 1.After DMA transfer for buffer 2 is finished, the DMA interrupt will occur and the DMA interrupt handler for audio driver will copies data from DMA buffer 2 to the buffers allocated by audio applications.

• Continue above steps for next DMA transfer cycle

Hardware Sample and Software Sample

• Hardware sample capability is determined by the hardware condition, such as for SGTL5000, hardware sample is 44.1 KHz, 2 channels and 16 bits data per sample

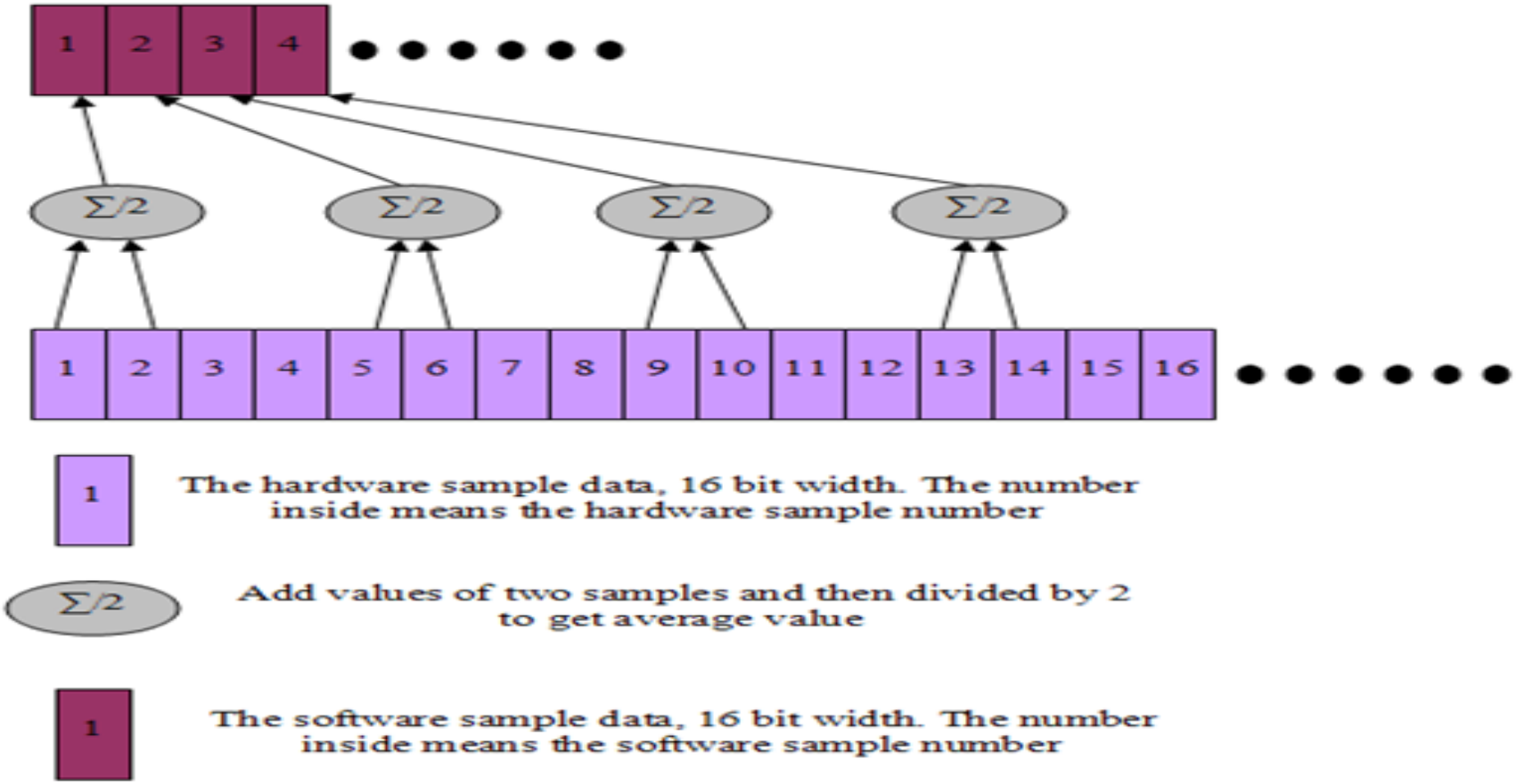
• software sample may have different requirements. For example, the audio application may set the software sample as 1 channel, 8K Hz sample rate and 16 bits data per sample.

• Audio driver should figure out the relationship between the hardware sample and software sample and implement the transition

• Example

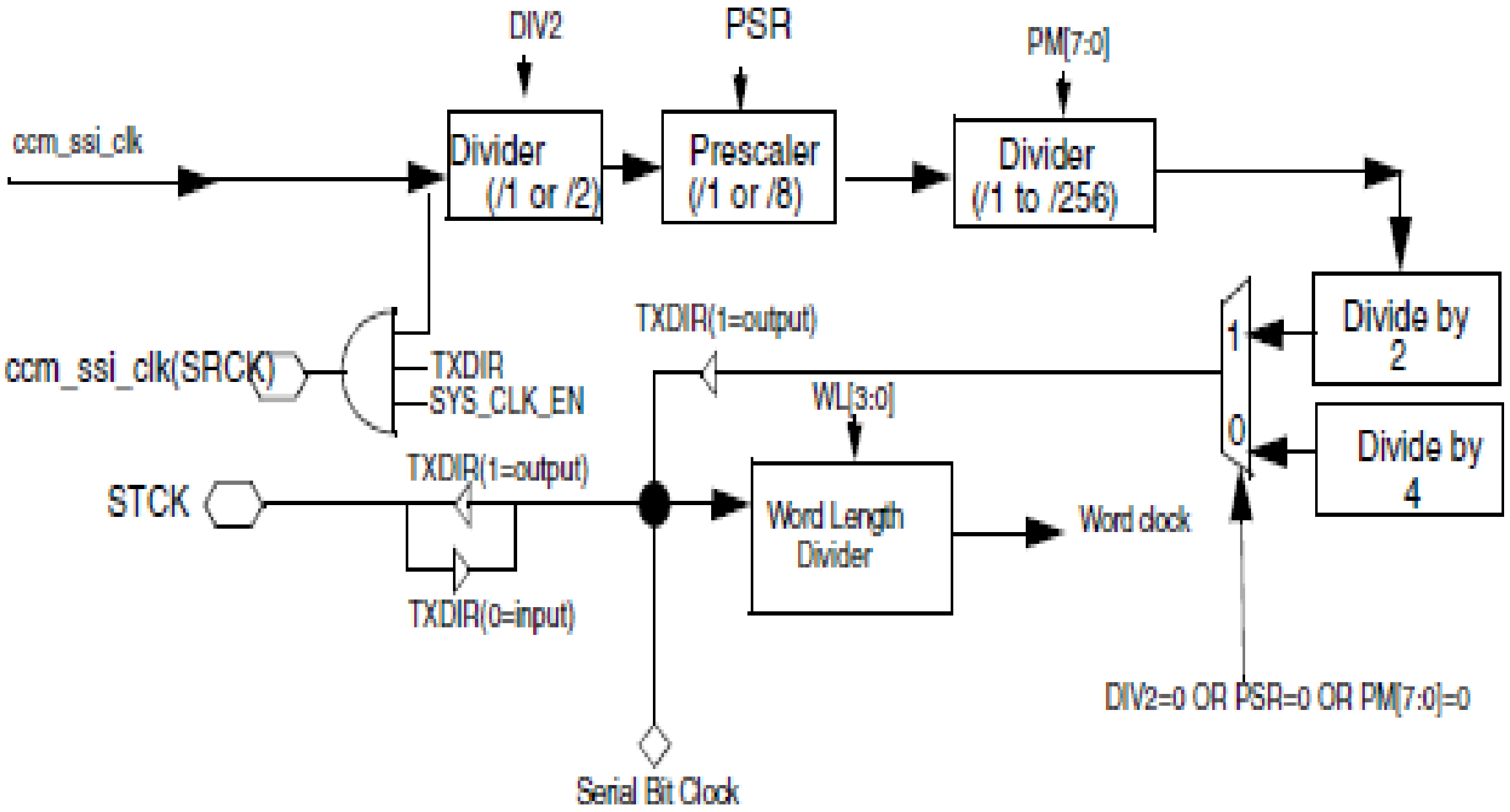
Consider the example mentioned above, that is, audio application requires the software sample as 1 channel, 22.05 KHz sample rate and 16 bits data per sample. Following figure shows how audio driver converts the hardware sample data to software sample data in this case

Hardware Sample and Software Sample



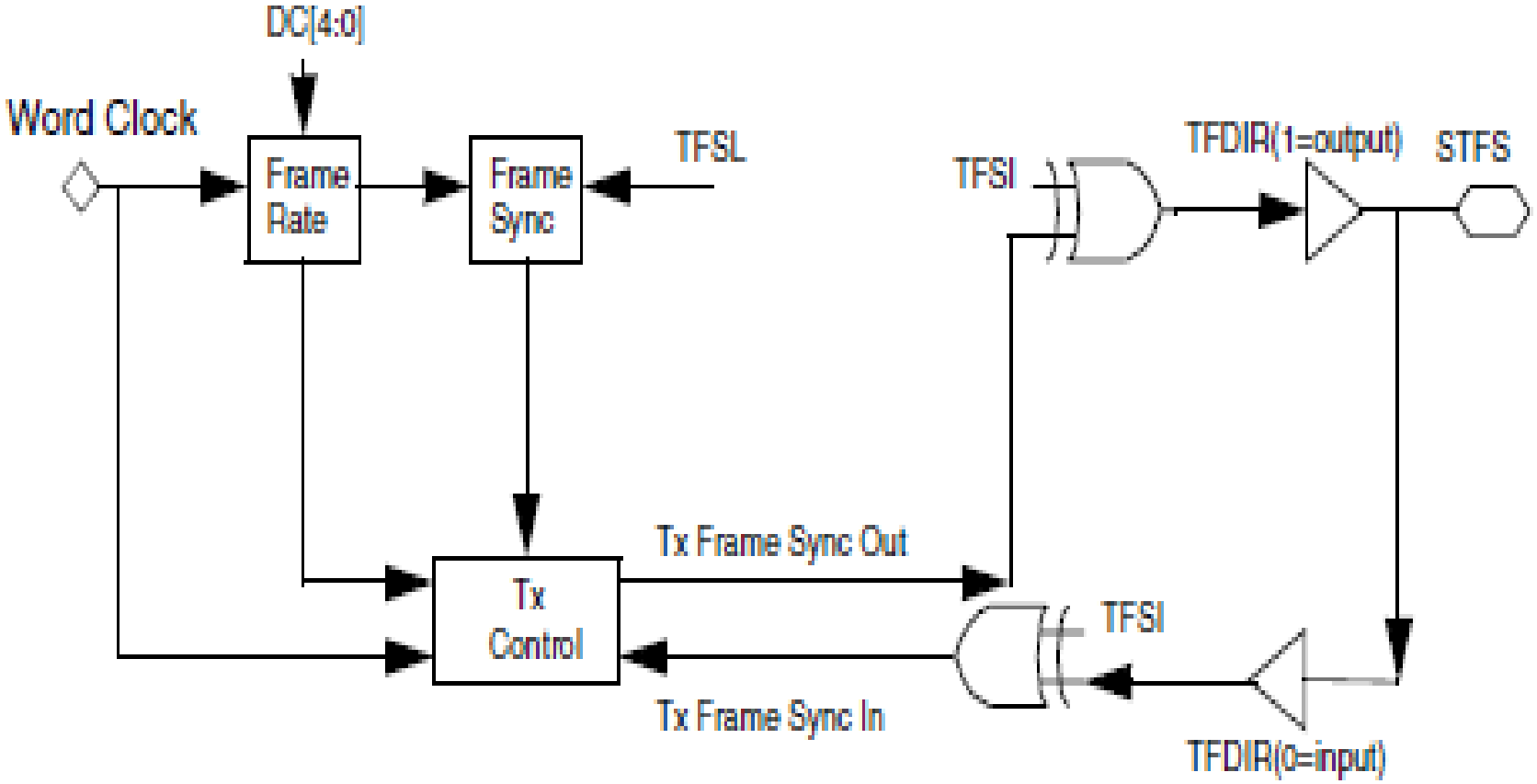
freescale semiconductor

SSI Bit Clock



freescale™ semiconductor

SSI FS Clock



freescale™ semiconductor

SSI Clock Equation

$$f_{INT\_BIT\_CLK} = f_{ccm\_ssi\_clk} / [(DIV2 + 1) \times (7 \times PSR + 1) \times (PM + 1) \times 2]$$

where PM=PM[7–0]

$$f_{FRAME\_SYN\_CLK} = (f_{INT\_BIT\_CLK}) / [(DC + 1) \times WL]$$

where DC = DC[4:0] and WL = 8, 10, 12, 16, 18, 20, 22, 24

Note: When DIV2 =0 and PSR = 0 and PM= 0, Frequency of bit_clk will be calculated as :-

$$f_{INT\_BIT\_CLK} = f_{ccm\_ssi\_clk} / 4$$

Figure 56-35. SSI Bit Clock Equation