Document Number: AN3067
Rev. 1, 02/2008

Interfacing the MSC711x TDM to A/D, D/A and Codecs

by Barbara Johnson
Digital Systems Division
Freescale Semiconductor, Inc.
Austin, TX

The time-division multiplexing (TDM) interface is a full-duplex serial port that allows MSC711x DSPs to communicate with a variety of serial devices, including industry-standard data converters, codecs, framers, other DSPs, and microprocessors. This application note presents examples of how to interface the MSC711x TDM port to A/D, D/A, and codec for various applications. A basic knowledge of the MSC711x TDM is assumed. The MSC711x reference manual and application notes are available at the web site listed on the back cover of this document.

Contents

1. TDM Overview 2
   1.1. TDM Architecture 3
   1.2. TDM Signals 4
   1.3. Clock and Frame Sync 6
   1.4. Frame Parameters 7
2. TDM Interface Examples 8
   2.1. Interfacing to an A/D Converter 8
   2.2. Interfacing to a D/A Converter 13
   2.3. Interfacing the TDM to a Codec 21

© Freescale Semiconductor, Inc., 2006, 2008. All rights reserved.

freescale™ semiconductor

1 TDM Overview

In a TDM system, two or more communications channels are transmitted over the same link with a different time slot allocated for each channel transmission. A channel can transmit data only during its allocated time slot. If a channel must send more data, it waits until its next assigned time slot is available. If a channel has no more data to send, the time slot is sent empty because it is still allocated to that channel. A periodic frame sync is generated to indicate the start of the frame so that the receiver can determine which channel is being received.

Figure 1 illustrates how a TDM works on an E1 line. A speech waveform is sampled 8000 times per second and each sample is represented as 8 bits, yielding a 64-Kbps channel capacity. Each frame represents one sample of 8 bits from each of the 32 channels. The frame is repeated every 125 ms. Therefore, the E1 interface yields an aggregate of:

$$32 \ channels \times 8 \ bits \ per \ channel \ / \ 125 \ ms = 2.048 \ Mbps$$

Eqn. 1

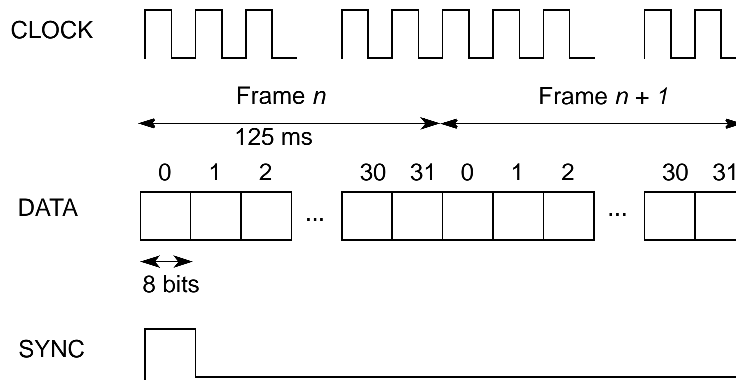


Figure 1. E1 Frame

The MSC711*x* TDM interface allows glueless connection to industry-standard E1/T1 lines, framers, data converters, codecs, other DSPs and microprocessors. The MSC711*x* has up to three identical TDM modules, depending on the device. Each TDM module has the following features:

- Independent or shared transmit and receive sections
- Independent or shared clock and frame sync
- Glueless interface to E1/T1 framers and MVIP, SCAS, and H.110 buses
- Hardware A-law/μ-law conversion
- Data rates up to 50 Mbps
- Up to 128 channels
- 8- or 16-bit word width
- Programmable direction for TDM clock and frame sync
- Programmable delay of 0–3 bits between frame sync signal and beginning of delay
- MSB or LSB first
- Loopback mode

1.1 TDM Architecture

The TDM has separate transmit and receive paths that can operate independently or share the clocking and synchronization. The transmit and receive clock and frame sync generators can generate the TDM transmit and receive serial bit clocks TDMxTCK/RCK. The TDM also supports an external source for the clock and frame sync. The TDM clock operates at up to 50 MHz with up to 128 channels of 8- or 16-bit data. The TDM frame sync signals TDMxTFS/RFS, which indicate the start of the frame, can also be input or output.

There is hardware support for A-law/μ-law conversion for 8-bit channel data. Data is converted to 13–14 bits and padded with zeros to create 16 bits of data.

The transmit data register, TDM*x*TDR, holds the data to be transmitted from the MSC711*x*. By default, the transmit FIFO is disabled and data in the transmit data register is transferred to the transmit shift register. Data is serially shifted out on the transmit data TDMxTD pin, one bit per transmit clock TDMxTCK.

A 4-line deep FIFO can optionally be enabled to buffer data from the transmit data register. Each line in the FIFO can hold one sample of 8/16-bits or packed 64-bits of data. In the latter case, the wide FIFO mode packs 8 × 8–bit or 4 × 16–bit samples in one FIFO line. Data from the FIFO is transferred to the transmit shift register, depending on the programmed watermark. If the amount of data in the FIFO is equal to the programmed watermark, transmission begins and data is transferred to the shift register and shifted out on the transmit data pin.

The receive path operates in the reverse sequence. Data on the receive data TDMxRD pin is shifted into the receive shift register, one bit per receive clock TDMxRCK cycle. If the receive FIFO is disabled, data in the receive shift register is transferred to the receive data register TDM*x*RDR. If the FIFO is enabled, the receive shift register fills the 4-line deep receive FIFO until the watermark level is reached. Data is then transferred to the receive data register.

To allow the SC1400 core to focus on data processing, the DMA can be configured to work with the TDM to transfer data in and out of the receive and transmit data registers. Two DMA channels are assigned to each TDM module for servicing the receive and transmit transfers. Figure 2 illustrates the architecture block diagram of a single TDM module.

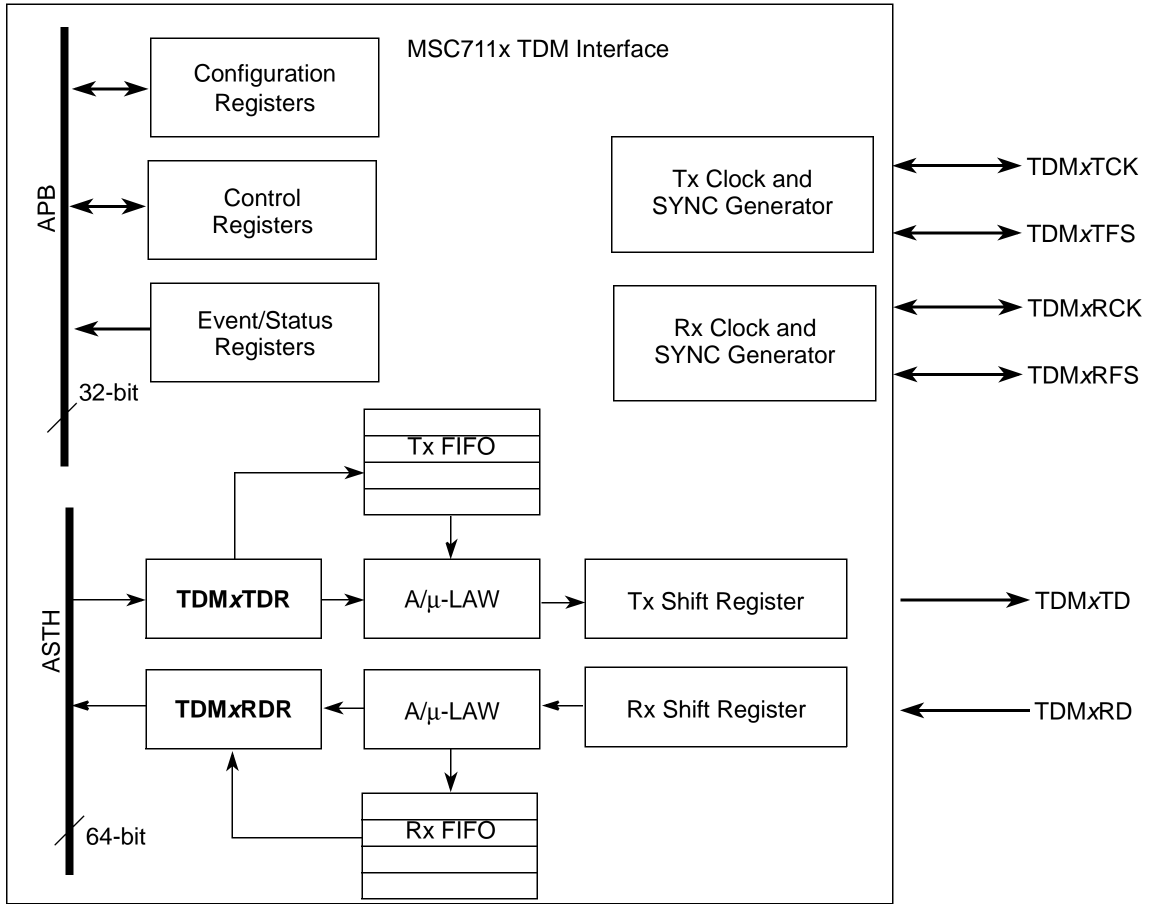


Figure 2. TDM Block Diagram

1.2 TDM Signals

Each TDM module has six signals, as shown in Table 1. Each of the transmit and receive sections has its own clock, frame sync, and data pins. The clock and frame sync can be configured as input or output because these signals are supplied from an external source or they are internally generated. The transmit data pin is output from the TDM and the receive data pin is input to the TDM.

Table 1. TDM Signals

Signal	Description	Input/Output
TDM*x*TCK	Serial bit clock for the transmit data.	I/O
TDM*x*RCK	Serial bit clock for the receive data.	I/O
TDM*x*TFS	Start of a transmit frame.	I/O
TDM*x*RFS	Start of a receive frame.	I/O
TDM*x*TD	Transmit data from the TDM.	O
TDM*x*RD	Receive data to the TDM.	I

The MSC711*x* exits reset with the I/O pins in software control mode, which selects GPIO instead of peripheral functionality. Therefore, the TDM functionality is not available until the TDM signals are configured in hardware control mode and the corresponding port pin registers are set up. The TDM pins should be initialized before the TDM registers are configured.

Table 2 shows the TDM pins and their GPIO port pin assignments. The TDM0 and TDM1 pins are configured using port A. The TDM2 pins are configured using both port A and port D. The port A and port D control registers (GPA_CTL and GPD_CTL) and the device configuration register (DEVCFG) must be configured as shown in Table 2 to enable the TDM pins.

Table 2. GPIO Pin Assignment for TDM

TDMx	Signal	Port	Bit	GPx_CTL	CHPCFG
TDM0	TDM0RCK	A	11	GPA_CTL[11] = 1	DEVCFG[PAS] = 0 or 1
	TDM0RFS	A	10	GPA_CTL[10] = 1	
	TDM0RD	A	9	GPA_CTL[9] = 1	
	TDM0TCK	A	8	GPA_CTL[8] = 1	
	TDM0TFS	A	7	GPA_CTL[7] = 1	
	TDM0TD	A	6	GPA_CTL[6] = 1	
TDM1	TDM1RCK	A	5	GPA_CTL[5] = 1	DEVCFG[PAS] = 0 or 1
	TDM1RFS	A	4	GPA_CTL[4] = 1	
	TDM1RD	A	3	GPA_CTL[3] = 1	
	TDM1TCK	A	2	GPA_CTL[2] = 1	
	TDM1TFS	A	1	GPA_CTL[1] = 1	
	TDM1TD	A	0	GPA_CTL[0] = 1	
TDM2	TDM2RCK	A	27	GPA_CTL[27] = 1	DEVCFG[PAS] = 1 DEVCFG[PDS] = 1
	TDM2RFS	D	4	GPD_CTL[4] = 1	
	TDM2RD	A	28	GPA_CTL[28] = 1	
	TDM2TCK	D	5	GPD_CTL[5] = 1	
	TDM2TFS	A	29	GPA_CTL[29] = 1	
	TDM2TD	D	6	GPD_CTL[6] = 1	

Example 1 shows how to configure the GPIO pins for TDM0, TDM1 and TDM2.

Example 1. GPIO Initialization for TDM0, TDM1 and TDM2

```
void InitGPIO()
{
GPIO *pstGPIO;
BTM *pstBTM;
pstGPIO = (GPIO *)(GPIO_BASE);
pstBTM = (BTM *)(BTM_BASE);
```

```
// Configure GPIO pins for TDM0 and TDM1 functionality
    pstGPIO->astPort[0].vuliPortControl = 0x00000FFF;
pstGPIO->astPort[0].vuliPortDataReg = 0;
pstGPIO->astPort[0].vuliPortDataDirReg = 0;

// Configure chip config to PAS & PDS for TDM2
pstBTM->vuliCHPCFG = pstBTM->vuliCHPCFG | 0x00000009;

// TDM2: T2TFS,T2RD,T2RCK
pstGPIO->astPort[0].vuliPortControl = pstGPIO->astPort[0].vuliPortControl | 0x38000000;

// TDM2: T2TD,T2TCK,T2RFS
pstGPIO->astPort[3].vuliPortControl = pstGPIO->astPort[3].vuliPortControl | 0x00000070;
}
```

1.3 Clock and Frame Sync

The MSC711*x* provides the flexibility to connect to different devices with different clock and frame sync requirements. Some devices internally generate the clock and frame sync from a master input clock, and others require these signals to be supplied from an external source. The TDM supports both input or output clock and frame sync, with independent or shared transmit and receive sections. In independent mode, the transmit and receive sections are clocked separately. In shared mode, they share the same clock and frame sync and only the transmit clock and frame sync pins are used. Table 3 summarizes the bit settings for the clock and frame sync operation.

Table 3. Clock and Frame Sync Settings

Tx/Rx	Clock and Frame Sync		Bit Settings	
Independent Tx/Rx	TDMxTCK clock	Input	TDMxTIR[TCOE] = 0	TDMxGIR[RTS] = 0
		Output	TDMxTIR[TCOE] = 1	
	TDMxTFS frame sync	Input	TDMxTIR[TSO] = 0	
		Output	TDMxTIR[TSO] = 1	
	TDMxRCK clock	Input	TDMxRIR[RCOE] = 0	
		Output	TDMxRIR[RCOE] = 1	
	TDMxRFS frame sync	Input	TDMxRIR[RSO] = 0	
		Output	TDMxRIR[RSO] = 1	
Shared Tx/Rx	TDMxTCK clock	Input	TDMxTIR[TCOE] = 0	TDMxGIR[RTS] = 1
		Output	TDMxTIR[TCOE] = 1	
	TDMxTFS frame sync	Input	TDMxTIR[TSO] = 0	
		Output	TDMxTIR[TSO] = 1	

When the bit clock is internally generated, the MSC711*x* timer B module provides the source for the TDM bit clock. Timer features such as the prescaler and compare registers can generate various frequencies. The input clock to the timer block can be derived from an external clock or from the APB clock, which operates at half the core frequency.

1.4 Frame Parameters

The MSC711*x* TDM has programmable frame parameters for compatibility with many devices. The TDM can receive or transmit up to 128 channels at a granularity of two channels. A single channel mode is also available for transmitting and receiving only one channel of data. Channel size is unified, so all channels in a frame are either 8 or 16 bits. The frame sync can be pulsed for one clock cycle or it can be asserted for the duration of one channel length. The frame sync phase is also programmable so that it is active at the logic one or logic zero state. Both the frame sync and data can be sampled or driven at the rising or falling edge of the clock. A programmable frame sync delay allows up to 3 cycles in half cycle increments of delay between the frame sync activation and the first data bit of the frame. Data can be received or transmitted with the most significant bit first (msb) or least significant bit first (lsb). Table 4 shows the frame parameters and associated registers.

Table 4. TDM Frame Parameters

Parameter	Tx/Rx	Register	Valid Settings
Number of channels	Transmit	TDMx_TFP[TNCF]	1, 2, 4, 6, ... 128 channels
	Receive	TDMx_RFP[RNCF]	
Channel size	Transmit	TDMx_TFP[TCS]	8 or 16 bits
	Receive	TDMx_RFP[RCS]	
Frame Sync Length	Transmit	TDMx_TIR[TSL]	1 bit or 1 word
	Receive	TDMx_RIR[RSL]	
Frame Sync Driven/Sampled	Transmit	TDMx_TIR[TFSE]	Rising or falling edge
	Receive	TDMx_RIR[RFSE]	
Frame Sync Active	Transmit	TDMx_TIR[TSA]	Logic 0 or logic 1
	Receive	TDMx_RIR[RSA]	
Frame Sync Delay	Transmit	TDMx_TIR[TFSD]	0 to 3 clocks in increments of 0.5 clocks
	Receive	TDMx_RIR[RFSD]	
Data Driven/Sampled	Transmit	TDMx_TIR[TDE]	Rising or falling edge
	Receive	TDMx_RIR[RDE]	
Data Order	Transmit	TDMx_TIR[TRDO]	msb first or lsb first
	Receive	TDMx_RIR[RRDO]	

2 TDM Interface Examples

This section provides examples of interfacing the TDM to an A/D, a D/A converter, and a codec for different types of applications. Testing was performed using the EVM711x Evaluation Module (EVM), which provides direct access to the TDM pins for easy connection to external devices. Software initialization of the TDM and DMA controller is also discussed.

2.1 Interfacing to an A/D Converter

The MSC711*x* TDM port can be used in data acquisition systems in which an A/D converter (ADC) receives a signal from an analog source such as a microphone, sensor, transducer, or function generator and converts the analog signal to digital data. The DSP then analyzes and processes the data and transmits it to the D/A converter (DAC), which converts the data back to an analog signal.

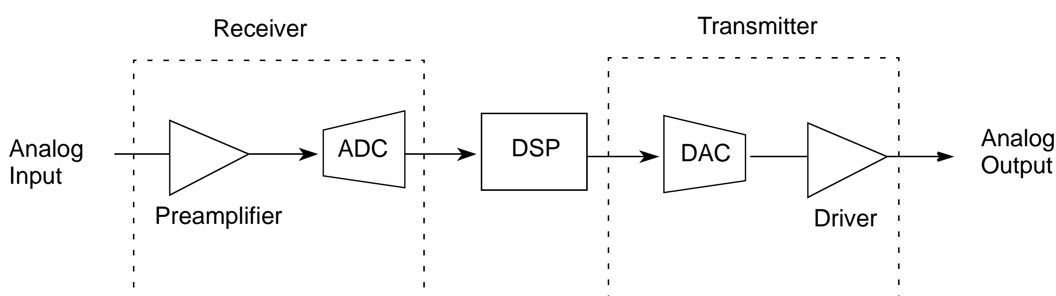


Figure 3. Example A/D and D/A System

In this case, the TDM0 module interfaces to the Burr-Brown ADS1602. The ADS1602 is a high-speed delta-sigma ADC for measurement applications in sonar, vibration analysis, and data acquisition. Although other ADC devices can interface to the TDM, the ADS1602 was selected for its high data rate and glueless connection to the TDM.

The ADS1602 requires an external clock of up to 40 MHz to be applied to the clock input CLK pin. With 16-bit samples, this means that the ADS1602 outputs data at rates up to:

$$40\ MHz\ /\ (16\ bits\ per\ sample) = 2.5\ Msamples/s \qquad \textbf{Eqn. 2}$$

The ADC outputs the serial clock, SCLK, to the TDM. Note that the SCLK and CLK operate at the same frequency of 40 MHz because the SCLK is internally generated by the ADC and is derived from the CLK signal. The SCLK connects to the TDM0_RCLK and clocks data into the MSC711*x*. The frame sync FSO is output from the ADC to indicate that data is available to be read from the ADC. The frame sync arrives every 2.5 MHz, as calculated in Equation 2. The FSO pin connects to the TDM0_RFS pin. The TDM receives data on the TDM0_RD pin from the ADC data output (DOUT) pin.

In addition to the standard TDM signals, a GPIO pin must connect from the DSP to the ADC synchronization control (SYNC) pin. The DSP must toggle the GPIO pin once after the ADC powers up to synchronize the ADC. After powerup, the GPIO is no longer needed. Figure 4 shows the ADC interface diagram.

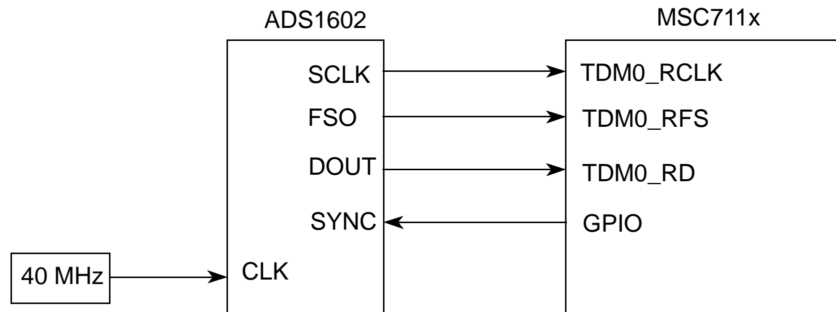


Figure 4. ADC Interface

Figure 5 shows the timing of the TDM signals as required by the ADS1602. The frame sync pulses every 16 clock cycles. In this case, only one channel of data is transferred every frame. The ADC asserts the frame sync for one bit clock. The frame sync is sampled on the falling edge of SCLK. The ADC drives the data on the rising edge of SCLK, with the msb driven first. There is a one clock cycle delay between the frame sync and the first data bit, so the first data bit sampled when the frame sync is active is the lsb (bit 0) of the previous frame. The msb (bit 15) of the current frame appears one cycle later.

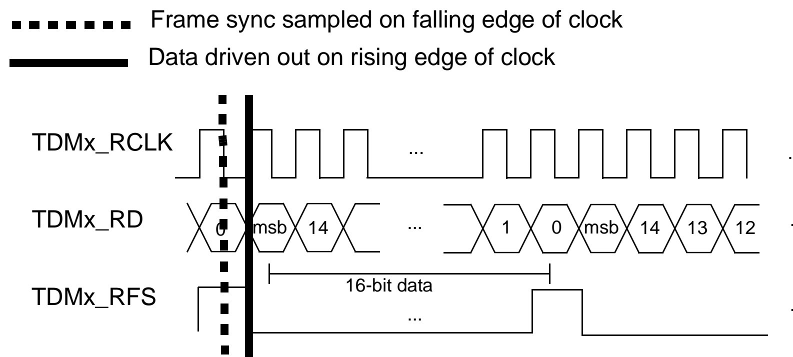


Figure 5. ADS1602 Serial Interface

2.1.1 TDM Initialization

Example 2 shows the TDM configuration based on the required ADC frame parameters. The receive and transmit sections of TDM0 are independent so that the receive and transmit operate with separate clock and frame sync signals. With these settings, the TDM0GIR register is cleared.

The TDM0RIR register is configured according to the ADS1602 interface requirements discussed previously. Both the clock and frame sync are input to the TDM0 from the ADC. The frame sync is sampled on the falling edge of the clock, and the data changes on the rising edge of the clock. A one bit sync delay occurs before the first data bit is sampled. Data is shifted in msb first.

Although it is not necessary, the receive FIFO is enabled in this example. The wide FIFO mode is not used, so each of the four FIFO lines holds one sample of 16-bit data. The DMA controller is enabled to transfer data from the TDM0RDREG to the receive data buffer.

The TDM0RFP is configured with the frame parameters. Each frame consists of one channel of 16-bit data.

The TDM0RCEN0 register is cleared so that no channel is active yet. The channel can be activated later, after the DMA settings are configured.

The TDM0RIER register is configured to enable interrupts when a receive frame sync error or a receive overrun event occurs.

Example 2. Configuring the TDM Parameters for the ADC

```
void InitTDMParams(void)
{
// TDM0 does not share signals with other TDMs
// Independent Rx/Tx
pstTDM0->vuliGIR = 0x00000000;

// Rx FIFO full with 1+ elements                               RFWM = 00
// Rx FIFO enabled                                              RFEN = 1
// Rx wide FIFO disabled                                        RWEN = 0
// Rx sync is input                                             RSO  = 0
// Rx sync is 1 bit                                             RSL  = 0
// Rx clock is input                                            RCOE = 0
// Rx DMA enabled                                               RDMA = 1
// Rx sync 1-bit delay                                         RFSD = 01
// Rx sync active on logic 1                                    RSA  = 0
// Rx data on rising edge clk                                   RDE  = 0
// Rx sync on falling edge clk                                  RFSE = 1
// Rx data MSB first                                            RRDO = 1
pstTDM0->vuliRIR = 0x00008053;                      //RFSD=01,RDE=0,RFSE=1

// 1 channel
// 16-bits/channel
pstTDM0->vuliRFP = 0x00000030;

// Deactivate all TDM channels
pstTDM0->avuliRCEN[0] = 0x00000000;
// Enable Rx Sync and Overrun Error Interrupt
pstTDM0->vuliRIER = 0x0000000c;
}
```

2.1.2 DMA Initialization

DMA channel 1 transfers data from the TDM0RDREG to a receive buffer when the receive FIFO is full. The DMA channel transfers one channel or 16 bits of data at a time from TDM0RDREG to the receive buffer. The source address specified in TCD0[SADDR] is programmed to be the TDM0RDREG address, and the destination address specified in TCD4[DADDR] is programmed to be the receive buffer address. Because the DMA controller accesses M1 memory beginning at 0x01800000, an offset of 0x01800000 must be added to the destination address if the buffer is located in M1 memory.

The TCD2[NBYTES] field specifies the inner minor loop byte count and is set to 2 bytes because each DMA transfer moves 16-bits of data. The source and destination sizes specified in TCD1[SSIZE] and TCD1[DSIZE] are programmed for 16 bits.

The source address offset specified in TCD1[SOFF] is set to 0 because TDM0RDREG is always the source of the transfer. However, the destination address offset specified in TCD4[DOFF] is set to 2 bytes because every transfer moves 2 bytes of data so the next destination address is incremented by 2 bytes for every transfer.

The TCD5[CITER] field specifies the number of major outer loop count. Each major iteration transfers 2 bytes of data. Therefore, to fill a buffer with SZ_BUF, the CITER field is set to SZ_BUF / 2 iterations. The beginning iteration count TCD7[BITER] field must be set to TCD5[CITER]. When the receive buffer is filled with SZ_BUF bytes of data, the DMA controller can be disabled as shown in the Example 3 or it can be configured to receive data continuously at the start of the receive buffer by setting the destination last address TCD6[DLAST] = -(SZ_BUF /2).

Example 3. Configuring the DMA Parameters for the ADC

```
void InitDMAParams()
{
// *************
// Receive side
// *************
// saddr = source addr is TDM Rx data register
// TDMRDREG = TDM0AHB_BASE + 0
pstDMA->astTCD[1].vuliSAddr = TDM0AHB_BASE;

// smod  =  0 source addr modulo disabled
// ssize = 16 bits source xfer size
// dmod  =  0 dest addr modulo disabled
// dsize = 16 bits dest xfer size
pstDMA->astTCD[1].vusiTransferAttr = 0x0101;
// soff = 0 source offset disabled
pstDMA->astTCD[1].vusiSOff = 0;
```

```
pstDMA->astTCD[1].vuliNBytes = 2;
// slast = 0 source last addr adjustment disabled
pstDMA->astTCD[1].vuliSLast = 0;
// daddr =  dest addr is Rx_buffer;
pstDMA->astTCD[1].vuliDAddr = (VUWord32)(&Rx_buffer);
pstDMA->astTCD[1].vusiCIter = SZ_BUF / 2;
pstDMA->astTCD[1].vusiDOff = 2;
pstDMA->astTCD[1].vuliDLastSGA = 0;
pstDMA->astTCD[1].vusiBIter = pstDMA->astTCD[1].vusiCIter;

// bwc          = 0
// major.linkch = 0
// major.e_link = 0
// e_sg                                                                    = 0
// d_req                                                                   = 0
// int_half                                                                = 0
// int_maj                                                                 = 0
// start                                                                   = 0
pstDMA->astTCD[1].vusiChannelCtrlStat = 0x0;
}
```

2.1.3 Starting the Transfer

The transfer can begin after the TDM and DMA parameters are set up as shown in Example 4. Clear the TDM and DMA error registers before beginning the transfer. Then enable DMA channel 1 to transfer data from the TDM0RDREG to the receive data buffer. Next, enable the receive section of the TDM and ensure that the frame sync is stable before activating the TDM channel. When the channel is enabled, the DMA controller begins to bring in data from the ADC to the receive data buffer and continues the transfers until the TCD5[CITER] decrements to zero. When the DMA transfer is complete, disable the receive section of the TDM.

Example 4. Starting the TDM Rx Transfer

```
void StartXfer(void)
{
// Clear TDM error registers
pstTDM0->vuliRER = 0x000001cc;
```

```
// Clear any DMA Errors
pstDMA->vucDMACERR = 0x40;

// Enable DMA chan 1 request
pstDMA->vuliDMAERQ = 0x00000002;

// Enable TDM Rx
pstTDM0->vuliRCR = 0x00000001;

    // Wait for sync
while(((pstTDM0->vuliRSR>>1)&3) != 2);

// Channel 0 is active
pstTDM0->avuliRCEN[0] = 0x01;

// Wait for Rx DONE = 1 for major loop completion
while((pstDMA->astTCD[1].vusiChannelCtrlStat & 0x0080) == 0);

// Disable TDM Rx
pstTDM0->vuliRCR = 0;
}
```

2.2 Interfacing to a D/A Converter

The Burr-Brown DAC8830 is a 16-bit DAC that is used in data acquisition systems, optical networking, and industrial process control applications. The DAC operates as a bus slave such that the DSP provides the serial clock TDM0_TCLK to the DAC SCLK input. For this particular DAC, the serial clock can operate at up to 50 MHz. The DSP accesses the DAC by driving the TDM0_TFS which is connected to the DAC chip select $\overline{CS}$ pin low. The 16-bit data from the DSP TDM0_TD pin is shifted out synchronously on the falling edge of the serial clock and latched on the rising edge of TDM0_TCLK into the input shift register, with msb first. After 16 clock cycles, the DSP deasserts $\overline{CS}$, which is connected to TDM0_TFS to indicate to the DAC to transfer the contents of its input register to the latch. At this time, the analog signal is output from the DAC. Figure 6 shows the DSP connection to the DAC.

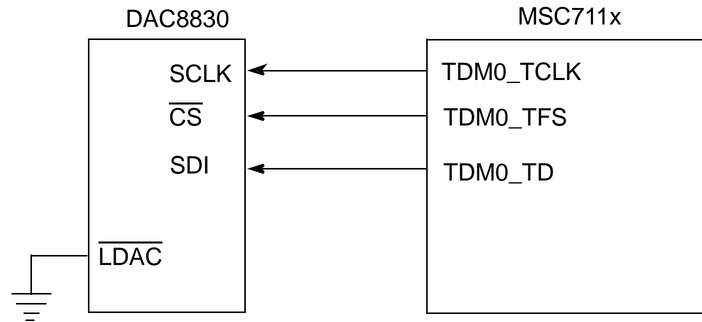
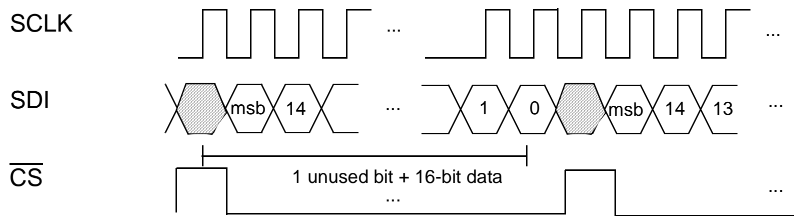


Figure 6. DAC Interface

The DAC requires the $\overline{CS}$ pin to be brought low for 16 clock cycles. If the $\overline{CS}$ is brought high before 16 bits are shifted in, data is corrupted and new data must be transferred to the DAC to be latched. The DAC requires that $\overline{CS}$ be high for a minimum 30 ns. With a 50 MHz clock, this means the $\overline{CS}$ must be high for at least 1.5 cycles. Data shifted into the DAC during the cycle when $\overline{CS}$ is high is thrown away and not used in the data conversion. The unused data bit is shown in the shaded area. This means that there are 17 clock cycles between frame syncs: 16 bits for valid data and one unused bit when the $\overline{CS}$ is high. The TDM does not support this interface where there are 17 clock cycles in a frame. To interface to this DAC, a workaround is needed. Figure 7 shows the DAC serial timing.



Note: An unused bit is not converted by DAC

Figure 7. DAC8830 Serial Interface

The MSC711*x* TDM requires the channel to be either 8 or 16 bits long. The channel size is global, so all channels in a frame must be the same size. The TDM interface does not support deactivation of individual bits in a channel. However, in a frame with two or more channels, a particular channel can be left inactive. This workaround would configure the TDM to have two 16-bit channels per frame so that the second 16-bits of data is discarded while $\overline{CS}$ is high. The first 16 bits of data is the actual data for conversion. Since the $\overline{CS}$ is high for the duration of the second channel, the minimum 30 ns requirement to keep $\overline{CS}$ high is met. Figure 8 shows the serial interface with two channels per frame, with the shaded bits discarded by the TDM. The resulting sampling rate is:

$$50\ MHz / (16\ bits\ per\ sample \times 2\ samples\ per\ frame) = 1.5625\ Msamples/s \qquad \text{Eqn. 3}$$

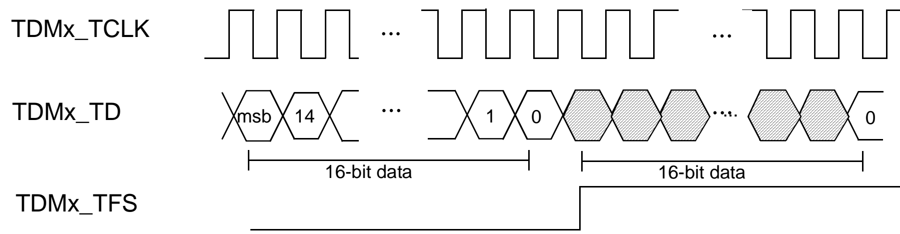


Figure 8. DAC8830 Serial Interface with 2 Channels

2.2.1 Clock and Frame Sync Generation

Since the DAC8830 requires the clock and frame sync to be input, the TDM can be configured to generate these signals from the timer B module. Timer features such as the prescaler and compare registers provide can generate various frequencies. The input clock to the timer block can be derived from an external clock or from the APB clock, which operates at half the core frequency. When the SC1400 core operates at 200 MHz, the desired TDM clock rate of 50 MHz can easily be derived from the APB clock of 100 MHz by selecting the APB as the source of the Timer clock.

With the source of the timer clock selected, the timer must be programmed to set up the count and compare parameters. The timer is configured to increment the counter for every rising edge of the primary count source, which is the 100 MHz clock from the APB. The counter counts up repeatedly until it reaches the compare value. The output flag OFLAG signal is toggled when the counter reaches the compare value. The OFLAG signal is output from the timer to provide the source for the TDM bit clock.

The timer compare register must be programmed with the number of rising edges of the timer input clock before the timer output is toggled. Since we need to generate a timer output of 50 MHz, the timer output must toggle at twice this frequency. Therefore, with a timer input clock of 100 MHz, the timer output must toggle at every rising edge of the timer input clock, as shown in Figure 9. The timer settings are summarized in Table 5.

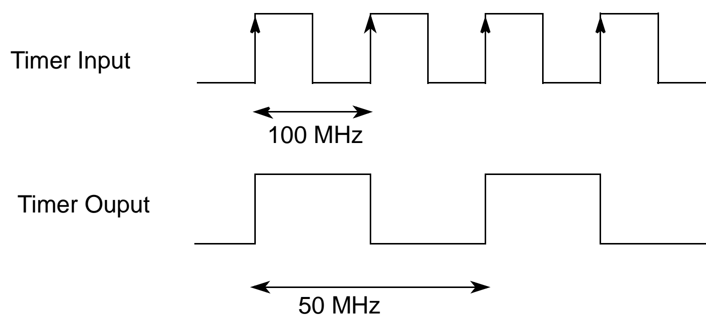


Figure 9. TDM Clock Generation

Table 5. Timer Parameters

Timer Parameter	Timer Settings	
Timer clock source	CLKCTL[TMUX] = 11	Timer clock source is APB clock
Count mode	TMRB0CTL[CM] = 001	Count rising edges of primary source
Primary count source	TMRB0CTL[PCS] = 1000	Prescaler (input clock divided by 1)
Count once	TMRB0CTL[ONCE] = 0	Count repeatedly
Count length	TMRB0CTL[LEN] = 1	Count until compare and then reinitialize
Count direction	TMRB0CTL[DIR] = 0	Count up
Output mode	TMRB0CTL[OM] = 011	Toggle OFLAG output on successful compare
Timer compare	TMRBCMP1 = 0	Compare to value of 1

Example 5 shows example code to generate the TDM clock from timer B.

Example 5. TDM Bit Clock Generation

```
void InitTDMClock()
{
TMRXn *pstTimerB;
CLK      *pstCLK;

pstCLK = (CLK *)(CLK_BASE);
pstTimerB = (TMRXn *)(TMRB_BASE);

// Source of Timer clock is APB clock
pstCLK->vuliCLKCTRL = pstCLK->vuliCLKCTRL | 0xC0000000;

// *************************************************************************
 // Generate 50 MHz TDM clock from Timer B
 //
    // COUNT MODE                 = 001  - Count rising edges of primary source
// PRIMARY COUNT SOURCE     = 1000 - Prescaler (Input clock divided by 1)
// SECONDARY SOURCE         = 00   - Counter 0 input signal (TIN0)
    // ONCE                       = 0    - Count repeatedly
    // LENGTH                     = 1    - Count till compare, then reinitialize
    // DIR                        = 0    - Count Up
    // EXT INIT                   = 0    - External counter/timers can not force a
//                                        re initialization of this counter/timer.
```

```
    // OUTPUT (OFLAG) MODE      = 011  - Toggle OFLAG output on successful compare
// ***************************************************************************

pstTimerB->vusiTMR_CTRL = 0x3023;
pstTimerB->vusiTMR_CMP1 = 0x0001;

// Set output enable
pstTimerB->vusiTMR_SCR = pstTimerB->vusiTMR_SCR | 0x01;
}
```

When the frame sync is internally generated, the TDM frame sync generator automatically generates the frame sync signal based on the TDM clock, the channel size, and the number of time slots in a frame. There is no need to program a register to set up the frame sync frequency. The frame sync generator divides the TDM clock by the size of the channel and by the number of time slots in a frame as shown in Figure 10. The output of this divider is the frame sync that marks the start of a frame.

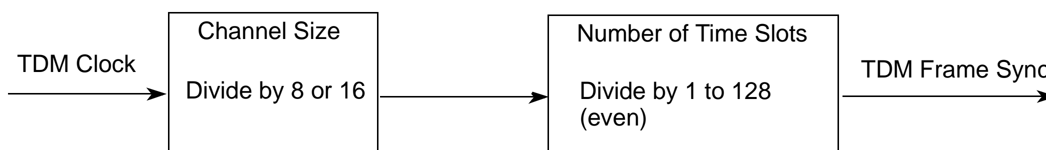


Figure 10. Frame Sync Generator

In this example, a TDM clock runs at 50 MHz, the TDM has two time slots of 16-bit data and the output frame sync is generated every 1.5625 MHz.

2.2.2 TDM Initialization

Example 6 shows the TDM configuration based on the required DAC frame parameters. The receive and transmit sections of TDM0 are independent so that the receive and transmit operate with separate clock and frame sync signals. With these settings, the TDM0GIR register is cleared.

The TDM0TIR register is configured according to the DAC8830 interface requirements but with two 16-bit channels. The first channel is converted and the second channel is discarded. The clock is output from the DSP to the DAC. The frame sync is internally generated by the TDM and input to the DAC. The frame sync is driven out on the rising edge of the clock, and the data changes on the rising edge of the clock. The frame sync is active on logic 0 for the duration of one channel. There is no delay between the frame sync and the first data bit. Data is shifted in msb first.

The transmit FIFO is enabled in this example. Each of the four FIFO lines holds one sample of 16-bit data. Data is loaded from the transmit data register (TDM0TDREG) to the FIFO and out to the transmit shift register. This example also enables the DMA controller to transfer data in from the transmit data buffer to TDM0TDREG.

The TDM0TFP register is configured for two channels of 16-bit data per frame.

The TDM0TCEN0 register is cleared so that no channel is active yet. The channel can be activated later after the DMA settings are configured.

The TDM0TCMA0 register is configured to mask the second channel of every frame. This channel gets discarded to meet the interface requirements of the DAC.

Example 6. Configuring the TDM Parameters for the DAC

```
The TDM0TIER register is configured to enable interrupts when a transmit frame sync error
or a transmit underrun event occurs.

void InitTDMParams(void)

{

// TDM0 does not share signals with other TDMs

// Independent Rx/Tx

pstTDM0->vuliGIR = 0x00000000;


// Tx FIFO full with 1+ elements                                          TFWM = 00

// Tx FIFO enabled                                                         TFEN = 1

// Tx wide FIFO disbled                                                    TWEN = 0

// Tx sync is output                                                       TSO  = 1

// Tx always out                                                           TAO  = 1

// Tx sync is 1 channel                                                    TSL  = 1

// Tx clock is input                                                       TCOE = 0

// Tx DMA enabled                                                          TDMA = 1

// Tx sync no delay                                                       TFSD = 00

// Tx sync active on logic 0                                               TSA  = 1

// Tx data on rising edge clk                                              TDE  = 0

// Tx sync on rising edge clk                                              TFSE = 0

// Tx data MSB first                                                       TRDO = 1

pstTDM0->vuliTIR = 0x0000A849;                                      //TFSE=0,TDE=0


// 2 channels
// 16-bits/channel

pstTDM0->vuliTFP = 0x00010030;
// Deactivate all TDM channels

pstTDM0->avuliTCEN[0] = 0x00000000;

// Masking enabled ch 1
pstTDM0->avuliTCMA[0] = 0x00000002;
```

```
// Enable Tx Sync & Underrun Error Interrupt
pstTDM0->vuliTIER = 0x0000000c;
}
```

2.2.3 DMA Initialization

DMA channel 0 is assigned to transfer data from a transmit data buffer to TDM0TDREG. The DMA setting for the transmit section is very similar to those for the receive section, except the source and destination addresses are different. The source address specified in TCD0[SADDR] is programmed to be the transmit buffer address and the destination address specified in TCD4[DADDR] is programmed to be the TDM0TDREG.

The destination address offset specified in TCD4[DOFF] is cleared to 0 because the TDM0TDREG is always the destination of the transfer. However, the source address offset specified in TCD1[SOFF] is set to 2 bytes because every transfer moves 2 bytes of data so that the next source address is incremented by 2 bytes. All other fields in the DMA transfer descriptor registers are programmed as discussed in the ADC example.

Example 7. Configuring the DMA Parameters for the DAC

```
void InitDMAParams()
{
// *************
// Transmit side
// *************
// saddr = source addr is Tx_buffer
pstDMA->astTCD[0].vuliSAddr = (VUWord32)(&Tx_buffer);

// smod  =  0 source addr modulo disabled
// ssize = 16 bits source xfer size
// dmod  =  0 dest addr modulo disabled
// dsize = 16 bits dest xfer size
pstDMA->astTCD[0].vusiTransferAttr = 0x0101;

pstDMA->astTCD[0].vusiSOff = 2;

// nbytes = 2 bytes
pstDMA->astTCD[0].vuliNBytes = 2;

pstDMA->astTCD[0].vuliSLast = -SZ_BUF;//continuous loop
//      pstDMA->astTCD[0].vuliSLast = 0;                             //single shot
```

```
// TDMTDREG = TDM0AHB_BASE + 8
pstDMA->astTCD[0].vuliDAddr = TDM0AHB_BASE + 8;

// total number bytes transferred = citer * nbytes
pstDMA->astTCD[0].vusiCIter = SZ_BUF / 2;

// doff = 0
pstDMA->astTCD[0].vusiDOff = 0;

// dlast_sga = 0;
pstDMA->astTCD[0].vuliDLastSGA = 0;

pstDMA->astTCD[0].vusiBIter = pstDMA->astTCD[0].vusiCIter;

// bwc          = 0
// major.linkch = 0
// major.e_link = 0
// e_sg                                                                  = 0
// d_req                                                                 = 0
// int_half                                                              = 0
// int_maj                                                               = 0
// start                                                                 = 0
pstDMA->astTCD[0].vusiChannelCtrlStat = 0x0000;
}
```

The TDM transmit transfer is started much like the receive transfer. The transfer can begin after the TDM and DMA parameters are set up as shown in Example 8. DMA channel 0 services the transmit side of the TDM. After the transmit section of the TDM is enabled, the sync should be stable before the TDM channel is activated. Only channel 0 is activated. The DMA begins to transmit data from the transmit data buffer to the DAC and continues the transfers until the TCD5[CITER] decrements to zero.

Example 8. Starting the TDM Tx Transfer

```
void StartXfer(void)
{
    // Clear TDM error registers
pstTDM0->vuliTER = 0x000001cc;
```

```
// Clear any DMA Errors
pstDMA->vucDMACERR = 0x40;

// Enable DMA chan 0 requests
pstDMA->vuliDMAERQ = 0x00000001;

// Enable TDM Tx
pstTDM0->vuliTCR = 0x00000001;

    // Wait for sync
while(((pstTDM0->vuliTSR>>1)&3) != 2);

// Channel 0 is active
pstTDM0->avuliTCEN[0] = 0x01; //ch 0 active

while(1);
}
```

2.3 Interfacing the TDM to a Codec

The MSC711*x* TDM port can also be used in consumer digital audio applications such as 16-bit 44.1 KHz sampling rate for CD-quality computer audio. A digital audio system using a codec as shown in Figure 11 provides an integrated A/D and D/A, which provides an input source such as a microphone. The DSP performs signal processing on the received data and transmits the processed data back to the codec, which outputs to a speaker or a headphone.

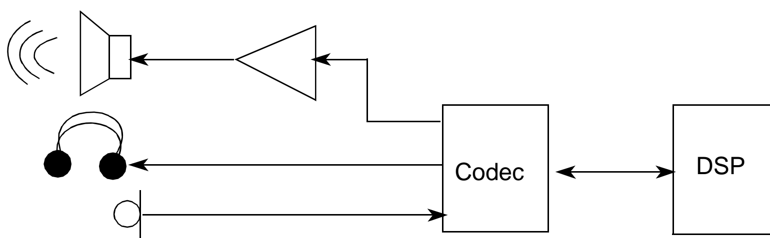


Figure 11. Digital Audio System

For interfacing the MSC711*x* TDM to a 16-bit stereo codec. The EVM711x uses the AKM AK4550 codec as shown in Figure 12 to input and output stereo channels to and from the DSP. The AK4550 is a 16-bit delta-sigma codec for portable digital audio systems. It supports sampling rates of up to 50 KHz.

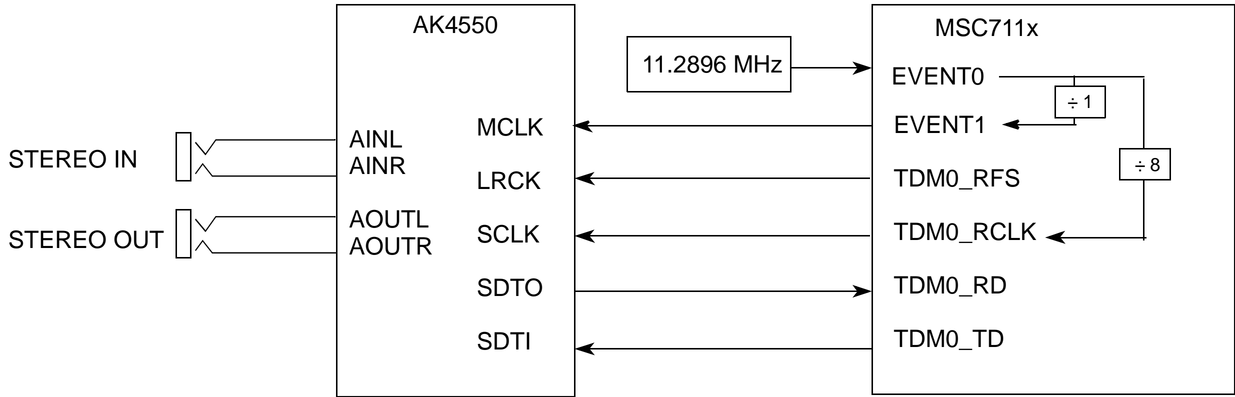


Figure 12. Codec Interface

2.3.1 Clock and Sync Generation

The AK4550 requires an external clock supplied to its master clock input MCLK pin. The MCLK frequency can be 256fs. With a sampling rate of 44.1 KHz, the required system input clock is:

$$MCLK = 256 \times 44.1\ KHz = 11.2896\ MHz$$

Eqn. 4

Therefore, a frequency of 11.2896 MHz must be supplied to the MCLK pin. Other sampling rates such as 32 KHz or 48 KHz can be used by changing the MCLK frequency to 8.192 MHz or 12.2880 MHz, respectively.

Although a clock oscillator can directly connect to the MCLK, the EVM711x connects the output of the 11.2896 MHz oscillator to the MSC711x EVENT0 pin. This connection gives the user the option of dividing the clock down and routing the scaled down clock to a second event pin, EVENT1. This routing scheme can use different clock oscillator frequencies. In this example, the optional division is bypassed so that the EVENT1 output is the same as the EVENT0 input. The code in Example 9 shows EVENT1 is driven by EVENT0.

Since the AK4550 requires the serial clock to be supplied to the serial clock input SCLK pin, the 11.2896 MHz clock input to the MSC711x can also be internally routed to the timer to generate the required serial clock and frame sync. Example 9 shows the timer input pin TIN0 driven by EVENT0.

Example 9. Event Port and Timer Settings

```
void InitEventPort()
{
        EV *pstEV;
        pstEV    = (EV *)(EV_BASE);

        // Mux0 input is EVNT0
        pstEV->astEVIN[0].vuliIN = pstEV->astEVIN[0].vuliIN | 0x00000080;
```

```
        // Mux0 output result to EVNT1 and TIN0
        pstEV->astEVOUT[0].vuliOUT = 0x78004238;
}
```

Note that the MCLK and SCLK do not operate at the same frequency. MCLK is 256fs and SCLK is 32fs. With a 44.1 KHz sampling rate, the serial input clock to the AK4550 is:

$$SCLK = 32 \times 44.1\ KHz = 1.4112\ MHz$$

Eqn. 5

The timer generates the TDM clock when it is output from the MSC711x. Because EVENT0 drives timer input TIN0 with 11.2896 MHz, this clock must be divided by 8 to obtain the 1.4112 MHz serial clock. Example 10 shows the code to configure the timer to generate the required clock.

The AK4550 also requires the frame sync to be supplied to the left/right clock LRCK pin from the MSC711x TDM0_RFS pin. When asserted, this pin indicates the left channel data and when deasserted, it indicates the right channel data. The frame sync is active for 16 clock cycles. The frame sync is asserted every 44.1 KHz:

$$LRCK = 1.4112\ MHz / (2\ channels \times 16\ bits\ per\ channel) = 44.1\ KHz$$

Eqn. 6

Example 10. Timer Setting

```
void InitTDMClockSync(void)
{
TMRXn    *pstTimerB;
CLK      *pstCLK;

pstCLK = (CLK *)(CLK_BASE);
pstTimerB = (TMRXn *)(TMRB_BASE);

// Source of Timer clock is APB clock
pstCLK->vuliCLKCTRL = pstCLK->vuliCLKCTRL | 0xC0000000;

// ***********************************************
    // COUNT MODE               = 001  - Count rising edges of primary source
    // PRIMARY COUNT SOURCE     = 0000 - Counter 0 input signal (TIN0)
    // SECONDARY SOURCE         = 00   - Counter 0 input signal (TIN0)
    // ONCE                     = 0    - Count repeatedly
```

```
    // LENGTH                    = 1    - Count till compare, then reinstalled
    // DIR                       = 0    - Count Up
    // EXT INIT                 = 0    - Ext ctr/tmrs can not force a reinit of this ctr/tmr
    // OUTPUT (OFLAG) MODE       = 011  - Toggle OFLAG output on successful compare
// ***********************************************
pstTimerB->vusiTMR_CTRL = 0x2023;                    //EVENT1 routing from EVENT0
pstTimerB->vusiTMR_CMP1 = 0x0007;                      //Divide by 8 from EVENT0
pstTimerB->vusiTMR_SCR = pstTimerB->vusiTMR_SCR | 0x01;      // Set output enable
}
```

2.3.2 TDM Initialization

Figure 12 shows the connection between the MSC711*x* and the codec. Data is shifted out of the AK4550 on the serial data output SDTO pin into the MSC711x TDM0_RD pin. Data is shifted into the codec on the serial data input SDTI pin from the MSC711x TDM0_TD pin.

Figure 13 shows the codec serial interface timing that applies to both the receive and transmit sections because they share the same clock and frame sync. The frame sync is driven out on the falling edge of the clock. Transmit data is driven and receive data is sampled on the falling edge of the clock. Each frame consists of the left and right channels, each 16-bits wide. When the frame sync is logic high, the left channel data is shifted in or out. When the frame sync is logic low, the right channel data is active. Example 11 shows the code to configure the TDM parameters for the codec.

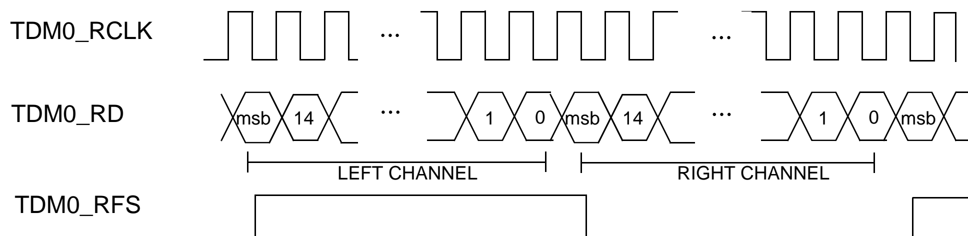


Figure 13. AK4550 Serial Interface

Example 11. Configuring the TDM Parameters for the Codec

```
void InitTDMParams(void)
{
TDM *pstTDM0;
pstTDM0 = (TDM *)(TDM0_BASE);

// TDM0 does not share signals with other TDMs
// Rx/Tx share same sync and clock
// LPBK=0, CTS=0, RTS=1
```

```
pstTDM0->vuliGIR = 0x00000001;
// Rx sync is 1-channel length
// Rx sync output driven on falling edge of clock
// Rx data sampled on falling edge of clock
pstTDM0->vuliRIR = 0x0002EA47;

// Tx sync is 1-channel length
// Tx sync output driven on falling edge of clock
// Tx data driven on falling edge of clock
pstTDM0->vuliTIR = 0x0002EA47;

// 2 channels
// 16-bits/channel
pstTDM0->vuliRFP = 0x00010030;

// 2 channels
// 16-bits/channel
pstTDM0->vuliTFP = 0x00010030;

// Deactivate all TDM channels
pstTDM0->avuliRCEN[0] = 0x00000000;
pstTDM0->avuliTCEN[0] = 0x00000000;

// Masking not enabled
pstTDM0->avuliTCMA[0] = 0x00000000;

// Enable Rx Sync & Overrun Error Interrupt
pstTDM0->vuliRIER = 0x0000000c;

// Enable Tx Sync & Underrun Error Interrupt
pstTDM0->vuliTIER = 0x0000000c;
}
```

2.3.3 DMA Initialization

The DMA configuration for the codec is similar to the previous examples except that the wide FIFO mode is used. Because the codec uses 16-bit data, the two channels are packed into each of the FIFO lines. Although the two channels occupy only 32 bits of the 64-bit wide FIFO line, each DMA transfer takes the entire 64 bits of data in wide FIFO mode. Unlike the previous examples, the number of bytes per TCD2[NBYTES] and the source and destination transfer sizes TCD1[SSIZE/DSIZE] are set to 8 bytes. The source and destination address offsets TCD1[SOFF] and TCD4[DOFF] must also be set to 8 bytes. Since each DMA transfer moves 8 bytes of data, TCD5[CITER] is set to SZ_BUF / 8 iterations to fill a buffer of size SZ_BUF.

All other fields in the DMA transfer descriptor registers are programmed as discussed in the previous examples.

Example 12. Configuring the DMA Parameters for the Codec

```
void InitDMAParams()
{
DMA *pstDMA;
pstDMA = (DMA *)(DMA_BASE);

// Group 0 Pri = 1 (highest); RR chan & fixed grp arbitrations
pstDMA->vuliDMACR  = 0x00000104;

pstDMA->DCHPRI[0]  = 0x1D;
pstDMA->DCHPRI[1]  = 0x1C;
pstDMA->DCHPRI[2]  = 0x82;
pstDMA->DCHPRI[3]  = 0x83;
pstDMA->DCHPRI[4]  = 0x04;
pstDMA->DCHPRI[5]  = 0x05;
pstDMA->DCHPRI[6]  = 0x0E;
pstDMA->DCHPRI[7]  = 0x0F;
pstDMA->DCHPRI[8]  = 0x08;
pstDMA->DCHPRI[9]  = 0x09;
pstDMA->DCHPRI[10] = 0x0a;
pstDMA->DCHPRI[11] = 0x0b;
pstDMA->DCHPRI[12] = 0x0c;
pstDMA->DCHPRI[13] = 0x0d;
pstDMA->DCHPRI[14] = 0x00;
pstDMA->DCHPRI[15] = 0x01;
```

```
// Disable all DMA Requests
pstDMA->vuliDMAERQ = 0x00000000;

// *************
// Receive side
// *************
// saddr = source addr is TDM Rx data register
// TDMRDREG = TDM0AHB_BASE + 0
pstDMA->astTCD[1].vuliSAddr = TDM0AHB_BASE;

// smod  =  0 source addr modulo disabled
// ssize = 64 bits source xfer size
// dmod  =  0 dest addr modulo disabled
// dsize = 64 bits dest xfer size
pstDMA->astTCD[1].vusiTransferAttr = 0x0303;

// soff = 0 source offset disabled
pstDMA->astTCD[1].vusiSOff = 0;

pstDMA->astTCD[1].vuliNBytes = 8;

// slast = 0 source last addr adjustment disabled
pstDMA->astTCD[1].vuliSLast = 0;

// daddr =  dest addr is Rx_buffer;
pstDMA->astTCD[1].vuliDAddr = (VUWord32)(&Rx_buffer);

pstDMA->astTCD[1].vusiCIter = SZ_BUF / 8;
pstDMA->astTCD[1].vusiDOff = 8;
pstDMA->astTCD[1].vuliDLastSGA = -SZ_BUF;
pstDMA->astTCD[1].vusiBIter = pstDMA->astTCD[1].vusiCIter;

// bwc          = 0
// major.linkch = 1 Link to channel 1
```

```
// major.e_link = 1
// e_sg                                                                     = 0
// d_req                                                                    = 0
// int_half                                                                 = 0
// int_maj                                                                  = 0
// start                                                                    = 0
pstDMA->astTCD[1].vusiChannelCtrlStat = 0x0120;

// *************
// Transmit side
// *************
// saddr = source addr is Tx_buffer
pstDMA->astTCD[0].vuliSAddr = (VUWord32)(&Rx_buffer);

// smod  =  0 source addr modulo disabled
// ssize = 64 bits source xfer size
// dmod  =  0 dest addr modulo disabled
// dsize = 64 bits dest xfer size
pstDMA->astTCD[0].vusiTransferAttr = 0x0303;

pstDMA->astTCD[0].vusiSOff = 8;

// nbytes = 8 bytes
pstDMA->astTCD[0].vuliNBytes = 8;

pstDMA->astTCD[0].vuliSLast = -SZ_BUF;

// TDMTDREG = TDM0AHB_BASE + 8
pstDMA->astTCD[0].vuliDAddr = TDM0AHB_BASE + 8;

// total number bytes transferred = citer * nbytes
pstDMA->astTCD[0].vusiCIter = SZ_BUF / 8;

// doff = 0
pstDMA->astTCD[0].vusiDOff = 0;
```

```
// dlast_sga = 0;
pstDMA->astTCD[0].vuliDLastSGA = 0;

pstDMA->astTCD[0].vusiBIter = pstDMA->astTCD[0].vusiCIter;

// bwc          = 0
// major.linkch = 0 Link to channel 0
// major.e_link = 1
// e_sg                                                                    = 0
// d_req                                                                   = 0
// int_half                                                                = 0
// int_maj                                                                 = 0
// start                                                                   = 0
pstDMA->astTCD[0].vusiChannelCtrlStat = 0x0020;
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
1-800-521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc., 2006, 2008.

Document Number: AN3067
Rev. 1
02/2008