TAS5026 & TAS5036 Pseudo Code Examples

Keith W. Engler *Digital Audio Solutions*

ABSTRACT

The TAS5026 and the TAS5036 are innovative, cost-effective, high performance 24-bit six-channel digital pulse width modulators (PWM) based on Equibit™ technology.

The purpose of this document is to supply a short description of how to communicate over IIC to each register in these devices.

The examples are in a “C” programming language format. They are considered pseudo-code since each function is just an example and not necessarily working code.

Table of Contents

Table of Figures

Topic **Page**

Table of Tables

Topic **Page**

1 Purpose

The purpose of this document is to supply pseudo-code methods to assist with software development of the TAS5026 and TAS5036 devices.

2 Disclaimer

The examples contained in this document are only examples. The accuracy of this information is neither expressed nor implied.

3 Description

This document will reference the June 2002 Data Manual for the “TAS5026 Six-Channel Digital Audio PWM Processor”.

This document will reference the November 2002 Data Manual for the “TAS5036 Six-Channel Digital Audio PWM Processor”.

This document is written in C-Code format. It has a Main that calls a function for each register. The function for each register is in a separate paragraph labeled accordingly.

4 I2C Protocol

Paragraph 2.5 in the TAS5026 Six-Channel Digital Audio PWM Processor Data Manual describes the I2C Serial Control Interface. The protocol is the same for the TAS5036.

There are two possible I2C Slave addresses. Externally changing the CSO pin 15 can change the address. If you are in Write mode then bit 0 should be set to zero. If you are in Read mode then bit 0 should be set to one.

The following table shows the value of the 8-bit data sent to the device in each mode.

Table 1: I2C Protocol

Read/Write Bit (Bit 0) 0 = Write 1 = Read	CSO Bit (Bit 1)	7 bit Slave Address (Binary)	Address Byte (Hex)	Address Byte (Binary)
0	0	0011010	0x34	00110100
1	0	0011010	0x35	00110101
0	1	0011011	0x36	00110110
1	1	0011011	0x37	00110111

5 IIC Pseudo-Code

This section defines functions for writing and reading to the IIC bus. This section introduces functions that are flexible enough to be used in any coding environment. There are 3 functions called that are specific to the IIC Driver application being implemented. They are StartCondition, StopCondition and SendByte.

5.1 Generic Write Functions

There section defines generic routines used to write IIC data.

***FUNCTION* SendRegisterData**

This function takes a single byte for the sub-address and a single byte as data. It writes this information over the IIC bus.

The function (GETIICAddr) returns the IIC Address. The definition of this function is not defined in this document.

```
void SendRegisterData(BYTE SubAddress,
                      BYTE RegData)
{
  const BYTE TotalBytes = 2;
  BYTE Data[TotalBytes] = {0};
  Data[0] = RegAddr;
  Data[1] = RegData;
  SendI2Cdata(GetIICAddr(), TotalBytes, Data);
}
```

***FUNCTION* SendIICData with Subaddress Embedded in Data**

This function takes a byte, an integer number and an array of bytes. The byte value contains the IIC Address, integer number represents the number of bytes to be sent over IIC and the array contains the bytes to be sent.

Error correction can be added to verify that the size of the array matches the number of bytes to be sent.

```
void SendI2Cdata(BYTE Address,
                 int ArrayCount,
                 BYTE DataArray[])
{
   StartCondition();
   SendByte(Address);
   for(int i=0; i<ArrayCount; i++)
   { SendByte(DataArray[i]); }
   StopCondition();
}
```

***FUNCTION* SendIICData with Subaddress Separate from Data**

This function takes two byte values, a SubAddr, an integer number and an array of bytes. The first byte is the IIC Address and the second byte value is the Register Address. The integer number represents the number of bytes to be sent over IIC and the array contains the bytes to be sent.

Error correction can be added to verify that the size of the array matches the number of bytes to be sent.

```
void SendI2CdataWithSubaddress(BYTE Address,
                               BYTE SubAddr,
                               int ArrayCount,
                               BYTE DataArray[])
{
  StartCondition();                  // Send Start Condition
  SendByte(Address &= 0xFE);  // Send I2C Slave Address
  SendByte(SubAddr);                 // Send sub-address 1C
  for(int i=0; i<ArrayCount; i++) // Send n bytes of data
    SendByte(DataArray[i]);
  StopCondition();                   // Send Stop Condition
}
```

5.2 Generic Read Functions

There section defines generic routines used to read IIC data.

***FUNCTION* ReadRegisterData**

This function takes a single byte for the IIC Address and a single byte as the sub-address. It writes this information over the IIC bus and receives a byte containing the data in that register.

```
BYTE ReadRegisterData(BYTE SlaveAddr,
                      BYTE RegAddr)
{
  const BYTE TotalBytes = 1;
  BYTE ReadData[TotalBytes];
  ReadData[0] = 0x00;
  ReadIICData(SlaveAddr, RegAddr, TotalBytes, ReadData);
  return ReadData[0];
}
```

***FUNCTION* ReadIICData**

This function takes a single byte for the IIC Address, a single byte as the sub-address, an integer defining how much data to return and an array of bytes to hold the returned data.

It writes this information over the IIC bus and receives an array of bytes containing the data in multiple sequential registers.

```
void ReadIICData(BYTE SlaveAddress,
                 BYTE SubAddress,
                 int Count,
                 BYTE * pBuff)
{
 BYTE DummyData[1];
 DummyData[0] = 0x00;
 SendI2CdataWithSubaddress (SlaveAddress, SubAddress, 0, DummyData);
 int i;
 int LastByte;

 for (i=0; i<Count; i++)
 {
  pBuff[i] = 0;  // zero the receive buffer
 }
 LastByte = (Count - 1);

 StartCondition();  // Send valid IIC Start Condition
 SendByte(SlaveAddress |= 0x01); // Send I2C Slave Address (Read)
 for (i=0; i<Count; i++)
 {
  pBuff[i] = ReceiveByte();
  if (i < LastByte)
    SendACK(); // Send an ACK when the byte is received
  else
    SendClock(); // Do not ACK the last byte received.
 }
 StopCondition();  // Send a valid IIC Stop Condition
}
```

6 Pseudo-Code for controlling multiple Registers

The functions below are examples of how to initialize and control multiple registers. This example uses Volume as an example but all registers can similarly be implemented.

6.1 TAS5026 and TAS5036 Initialization

***FUNCTION* Init**

This function is a single point where the initialization of each register can take place.

This function calls InitVolume but the user can create their required initialization routines. Since the TAS5026 and TAS5036 default conditions are already optimized, the volume may be the only registers that need to be set.

```
function Init()
{
  //Set Volume data per Appendix A in the Data Manual where 0 db = 0xC9
  InitVolume (0xC9, 0xC9, 0xC9, 0xC9, 0xC9, 0xC9)
}
```

***FUNCTION* InitVolume**

This function takes a byte for each of the six channels in the device.

The byte for each channel is sent to the function that controls the respective volume registers.

```
function InitVolume(BYTE Ch1Data,
                    BYTE Ch2Data,
                    BYTE Ch3Data,
                    BYTE Ch4Data,
                    BYTE Ch5Data,
                    BYTE Ch6Data,
                    )
{
  SetVolumeControlReg1Data (Ch1Data);  // described later in this document
  SetVolumeControlReg2Data (Ch2Data);  // described later in this document
  SetVolumeControlReg3Data (Ch3Data);  // described later in this document
  SetVolumeControlReg4Data (Ch4Data);  // described later in this document
  SetVolumeControlReg5Data (Ch5Data);  // described later in this document
  SetVolumeControlReg6Data (Ch6Data);  // described later in this document
}
```

6.2 Master Volume Example

The TAS5026 and TAS5036 do not explicitly have a Master Volume but one can be created in software. In this example, each Channel Volume is treated as an offset from the Master Volume and is changed relative to the Master Volume. The value for each Channel Volume can be saved globally and accessed when the Master Volume function is called. The volume offset is always positive, therefore, it can only add to the Master Volume Control.

Figure 1 is an example of a volume control panel. The Figure 2 is an example of the offset panel for controlling the volume offset for each channel from the master volume.

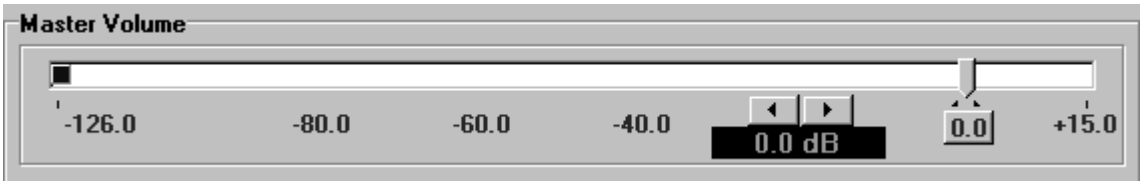

Figure 1: Master Volume Control Panel

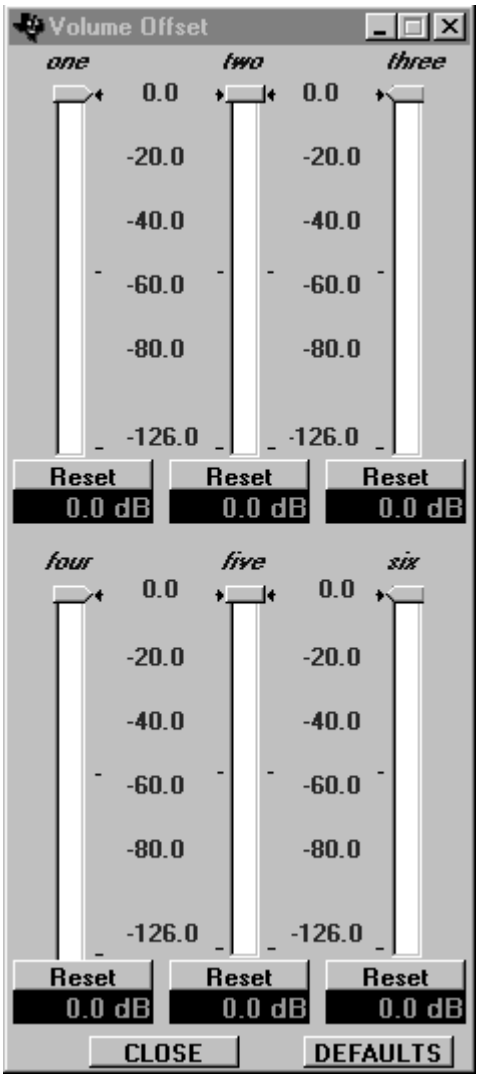

Figure 2: Volume Offset Control Panel

***FUNCTION* MasterVolume**

This function takes a byte which represents the value for the Master Volume.

The absolute volume is the result of the Volume Offset plus the Master Volume. The result is sent to the volume register after verifying it is within range.

```
function MasterVolume(BYTE MasterVolume){
  Ch1Offset = (MasterVolume + GetVolumeOffset(1);
  Ch2Offset = (MasterVolume + GetVolumeOffset(2);
  Ch3Offset = (MasterVolume + GetVolumeOffset(3);
  Ch4Offset = (MasterVolume + GetVolumeOffset(4);
  Ch5Offset = (MasterVolume + GetVolumeOffset(5);
  Ch6Offset = (MasterVolume + GetVolumeOffset(6);
  If (Ch1Offset <MinVolume) Ch1Offset = MinVolume;
  If (Ch2Offset <MinVolume) Ch2Offset = MinVolume;
  If (Ch3Offset <MinVolume) Ch3Offset = MinVolume;
  If (Ch4Offset <MinVolume) Ch4Offset = MinVolume;
  If (Ch5Offset <MinVolume) Ch5Offset = MinVolume;
  If (Ch5Offset <MinVolume) Ch6Offset = MinVolume;
  SetVolumeControlReg1Data (Ch1Offset);  // described later in this document
  SetVolumeControlReg2Data (Ch2Offset);  // described later in this document
  SetVolumeControlReg3Data (Ch3Offset);  // described later in this document
  SetVolumeControlReg4Data (Ch4Offset);  // described later in this document
  SetVolumeControlReg5Data (Ch5Offset);  // described later in this document
  SetVolumeControlReg6Data (Ch6Offset);  // described later in this document
}
```

***FUNCTION* SetVolumeOffset**

This function takes an integer value representing the channel number and a byte value which represents the value for the Master Volume.

The intention of this function is to set the CH1Offset value. This value can be a globally defined variable.

```
function SetVolumeOffset (int ChannelNumber,
                          BYTE ChannelValue)
{
 if (ChannelNumber == 1)
   Ch1Offset = ChannelValue;
 else
   if (ChannelNumber == 2)
     Ch2Offset = ChannelValue;
   else
     if (ChannelNumber == 3)
       Ch3Offset = ChannelValue;
     else
       if (ChannelNumber == 4)
         Ch4Offset = ChannelValue;
       else
         if (ChannelNumber == 5)
           Ch5Offset = ChannelValue;
         else
           if (ChannelNumber == 6)
             Ch6Offset = ChannelValue;
}
```

***FUNCTION* GetVolumeOffset**

This function takes an integer value representing the channel number and returns a byte value which represents the value for the Volume Offset.

```
BYTE ChannelOffsetValue function GetVolumeOffset(int ChannelNumber)
{
  if (ChannelNumber == 1)
    return Ch1Offset;
  else
    if (ChannelNumber == 2)
      return Ch2Offset;
    else
      if (ChannelNumber == 3)
        return Ch3Offset;
      else
        if (ChannelNumber == 4)
          return Ch4Offset;
        else
          if (ChannelNumber == 5)
            return Ch5Offset;
          else
            if (ChannelNumber == 6)
              return Ch6Offset;
}
```

7 Pseudo-Code for controlling individual Registers

All functions below will assume that the CSO bit is set to 0.

7.1 General Status Register (Hex Addr 0x00)

This is a Read Only Register. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetStatusRegisterData**

This function returns a byte value which represents the value of the Status register.

```
Byte function GetStatusRegisterData()
{
  SlaveAddr = 0x34;
  RegAddr = 0x00;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

7.2 Error Status Register (Hex Addr 0x01)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register

***FUNCTION* SetErrorStatusRegisterData**

This function allows the user to clear the Error Status register.

```
function SetErrorStatusRegisterData()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x00; //Set data per table in Section 3 in Data Manual
                    // default = 0x00
  SendRegisterData(SlaveAddr, Data);
}
```

***FUNCTION* GetErrorStatusRegisterData**

This function returns a byte value which represents the value of the Error Status register.

```
Byte function GetErrorStatusRegisterData()
{
  SlaveAddr = 0x34;
  RegAddr = 0x01;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

7.3 System Control Register 0 (Hex Addr 0x02)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register

***FUNCTION* GetSysControlReg0Data**

This function returns a byte value which represents the value of the Control 0 Register.

```
Byte function GetSysControlReg0Data()
{
  SlaveAddr = 0x34;
  RegAddr = 0x02;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetSysControlReg0Data**

This function sets the value of the Sys Control 0 Register.

```
function SetSysControlReg0Data()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                    // default = 0x05
  SendRegisterData(SlaveAddr, Data);
}
```

7.4 System Control Register 1 (Hex Addr 0x03)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetSysControlReg1Data**

This function returns a byte value which represents the value of the Sys Control 1 Register.

```
Byte function GetSysControlReg1Data()
{
  SlaveAddr = 0x34;
  RegAddr = 0x03;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetSysControlReg1Data**

This function sets the value of the Sys Control 1 Register.

```
function SetSysControlReg1Data()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x7E
  SendRegisterData(SlaveAddr, Data);
}
```

7.5 Error Recovery Register (Hex Addr 0x04)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetErrorRecoverRegData**

This function returns the value of the Error Status Recovery Register.

```
Byte function GetErrorRecoverRegData()
{
  SlaveAddr = 0x34;
  RegAddr = 0x04;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetErrorRecoverRegData**

This function sets the value of the Error Status Recovery Register.

```
function SetErrorRecoverRegData ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                  // default = 0xFF
  SendRegisterData(SlaveAddr, Data);
}
```

7.6 Automute Delay Register (Hex Addr 0x05)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetAutomuteDelayRegData**

This function returns the value of the Auto Mute Delay Register.

```
Byte function GetAutomuteDelayRegData()
{
  SlaveAddr = 0x34;
  RegAddr = 0x05;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetAutomuteDelayRegData**

This function sets the value of the Auto Mute Delay Register.

```
function SetAutomuteDelayRegData ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x00
  SendRegisterData(SlaveAddr, Data);
}
```

7.7 DC Offset Control Register Channel 1 (Hex Addr 0x06)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetOffsetControlReg1Data**

This function returns the value of the Offset Control 1Register.

```
Byte function GetOffsetControlReg1Data()
{
  SlaveAddr = 0x34;
  RegAddr = 0x06;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetOffsetControlReg1Data**

This function sets the value of the Offset Control 1Register.

```
function SetOffsetControlReg1Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x00
  SendRegisterData(SlaveAddr, Data);
}
```

7.8 DC Offset Control Register Channel 2 (Hex Addr 0x07)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetOffsetControlReg2Data**

This function returns the value of the Offset Control 2Register

```
Byte function GetOffsetControlReg2Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x07;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetOffsetControlReg2Data**

This function sets the value of the Offset Control 2Register.

```
function SetOffsetControlReg2Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x00
  SendRegisterData(SlaveAddr, Data);
}
```

7.9 DC Offset Control Register Channel 3 (Hex Addr 0x08)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetOffsetControlReg3Data**

This function returns the value of the Offset Control 3Register.

```
Byte function GetOffsetControlReg3Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x08;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetOffsetControlReg3Data**

This function sets the value of the Offset Control 3Register.

```
function SetOffsetControlReg3Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x00
  SendRegisterData(SlaveAddr, Data);
}
```

7.10 DC Offset Control Register Channel 4 (Hex Addr 0x09)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetOffsetControlReg4Data**

This function returns the value of the Offset Control 4Register.

```
Byte function GetOffsetControlReg4Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x09;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetOffsetControlReg4Data**

This function sets the value of the Offset Control 4Register.

```
function SetOffsetControlReg4Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x00
  SendRegisterData(SlaveAddr, Data);
}
```

7.11 DC Offset Control Register Channel 5 (Hex Addr 0x0A)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetOffsetControlReg5Data**

This function returns the value of the Offset Control 5Register.

```
Byte function GetOffsetControlReg5Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x0A;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetOffsetControlReg5Data**

This function sets the value of the Offset Control 5Register.

```
function SetOffsetControlReg5Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x00
  SendRegisterData(SlaveAddr, Data);
}
```

7.12 DC Offset Control Register Channel 6 (Hex Addr 0x0B)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetOffsetControlReg6Data**

This function returns the value of the Offset Control 6Register.

```
Byte function GetOffsetControlReg6Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x0B;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetOffsetControlReg6Data**

This function sets the value of the Offset Control 6Register.

```
function SetOffsetControlReg6Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x00
  SendRegisterData(SlaveAddr, Data);
}
```

7.13 Interchannel Delay Register Channel 1 (Hex Addr 0x0C)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetInterchannelDelay1Data**

This function returns the value of the Inter-Channel Delay 1Register

```
Byte function GetInterchannelDelay1Data()
{
  SlaveAddr = 0x34;
  RegAddr = 0x0C;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetInterchannelDelay1Data**

This function sets the value of the Inter-Channel Delay 1Register.

```
function SetInterchannelDelay1Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x00
  SendRegisterData(SlaveAddr, Data);
}
```

7.14 Interchannel Delay Register Channel 2 (Hex Addr 0x0D)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetInterchannelDelay2Data**

This function returns the value of the Inter-Channel Delay 2Register

```
Byte function GetInterchannelDelay2Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x0D;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetInterchannelDelay2Data**

This function sets the value of the Inter-Channel Delay 2Register.

```
function SetInterchannelDelay2Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x10
  SendRegisterData(SlaveAddr, Data);
}
```

7.15 Interchannel Delay Register Channel 3 (Hex Addr 0x0E)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetInterchannelDelay3Data**

This function returns the value of the Inter-Channel Delay 3Register

```
Byte function GetInterchannelDelay3Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x0E;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetInterchannelDelay3Data**

This function sets the value of the Inter-Channel Delay 3Register.

```
function SetInterchannelDelay3Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                  // default = 0x20
  SendRegisterData(SlaveAddr, Data);
}
```

7.16 Interchannel Delay Register Channel 4 (Hex Addr 0x0F)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetInterchannelDelay4Data**

This function returns the value of the Inter-Channel Delay 4Register.

```
Byte function GetInterchannelDelay4Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x0F;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetInterchannelDelay4Data**

This function sets the value of the Inter-Channel Delay 4Register.

```
function SetInterchannelDelay4Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x30
  SendRegisterData(SlaveAddr, Data);
}
```

7.17 Interchannel Delay Register Channel 5 (Hex Addr 0x10)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetInterchannelDelay5Data**

This function returns the value of the Inter-Channel Delay 5Register

```
Byte function GetInterchannelDelay5Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x10;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetInterchannelDelay5Data**

This function sets the value of the Inter-Channel Delay 5Register.

```
function SetInterchannelDelay5Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x40
  SendRegisterData(SlaveAddr, Data);
}
```

7.18 Interchannel Delay Register Channel 6 (Hex Addr 0x11)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetInterchannelDelay6Data**

This function returns the value of the Inter-Channel Delay 6Register.

```
Byte function GetInterchannelDelay6Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x11;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetInterchannelDelay6Data**

This function sets the value of the Inter-Channel Delay 6Register.

```
function SetInterchannelDelay6Data ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // default = 0x50
  SendRegisterData(SlaveAddr, Data);
}
```

7.19 ABD Delay Register (Hex Addr 0x12)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

This feature only applies to the TAS5036.

***FUNCTION* GetABDDelayData**

This function returns the value of the ABD Delay Register.

```
Byte function GetABDDelayData ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x12;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetABDDelayData**

This function sets the value of the ABD Delay Register.

```
function SetABDDelayData ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                  // default = 0x07
  SendRegisterData(SlaveAddr, Data);
}
```

7.20 Volume Control Register Channel 1 (Hex Addr 0x13)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetVolumeControlReg1Data**

This function returns the Volume Control 1 Register value.

```
Byte function GetVolumeControlReg1Data()
{
  SlaveAddr = 0x34;
  RegAddr = 0x13;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetVolumeControlReg1Data**

This function takes a BYTE and sets the Volume Control Register to that value.

```
function SetVolumeControlReg1Data (BYTE DataIn)
{
  SlaveAddr = 0x34;
  BYTE Data = DataIn //Set data per Appendix A in Data Manual
                                    // 0 db = 0xC9
  SendRegisterData(SlaveAddr, Data);
}
```

7.21 Volume Control Register Channel 2 (Hex Addr 0x14)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetVolumeControlReg2Data**

This function returns the Volume Control 2 Register value.

```
Byte function GetVolumeControlReg2Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x14;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetVolumeControlReg2Data**

This function takes a BYTE and sets the Volume Control Register to that value.

```
function SetVolumeControlReg2Data (BYTE DataIn)
{
  SlaveAddr = 0x34;
  BYTE Data = DataIn; //Set data per Appendix A in Data Manual
                                    // 0 db = 0xC9
  SendRegisterData(SlaveAddr, Data);
}
```

7.22 Volume Control Register Channel 3 (Hex Addr 0x15)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetVolumeControlReg3Data**

This function returns the Volume Control 3 Register value

```
Byte function GetVolumeControlReg3Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x15;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetVolumeControlReg3Data**

This function takes a BYTE and sets the Volume Control Register to that value.

```
function SetVolumeControlReg3Data (BYTE DataIn)
{
  SlaveAddr = 0x34;
  BYTE Data = DataIn; //Set data per Appendix A in Data Manual
                                    // 0 db = 0xC9
  SendRegisterData(SlaveAddr, Data);
}
```

7.23 Volume Control Register Channel 4 (Hex Addr 0x16)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetVolumeControlReg4Data**

This function returns the Volume Control 4 Register value

```
Byte function GetVolumeControlReg4Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x16;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetVolumeControlReg4Data**

This function takes a BYTE and sets the Volume Control Register to that value.

```
function SetVolumeControlReg4Data (BYTE DataIn)
{
  SlaveAddr = 0x34;
  BYTE Data = DataIn; //Set data per Appendix A in Data Manual
                                    // 0 db = 0xC9
  SendRegisterData(SlaveAddr, Data);
}
```

7.24 Volume Control Register Channel 5 (Hex Addr 0x17)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetVolumeControlReg5Data**

This function returns the Volume Control 5 Register value

```
Byte function GetVolumeControlReg5Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x17;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetVolumeControlReg5Data**

This function takes a BYTE and sets the Volume Control Register to that value.

```
function SetVolumeControlReg5Data (BYTE DataIn)
{
  SlaveAddr = 0x34;
  BYTE Data = DataIn; //Set data per Appendix A in Data Manual
                                    // 0 db = 0xC9
  SendRegisterData(SlaveAddr, Data);
}
```

7.25 Volume Control Register Channel 6 (Hex Addr 0x18)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetVolumeControlReg6Data**

This function returns the Volume Control 6 Register value.

```
Byte function GetVolumeControlReg6Data ()
{
  SlaveAddr = 0x34;
  RegAddr = 0x18;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetVolumeControlReg6Data**

This function takes a BYTE and sets the Volume Control Register to that value.

```
function SetVolumeControlReg6Data (BYTE DataIn)
{
  SlaveAddr = 0x34;
  BYTE Data = DataIn; //Set data per Appendix A in Data Manual
                                    // 0 db = 0xC9
  SendRegisterData(SlaveAddr, Data);
}
```

7.26 Individual Channel Mute (Hex Addr 0x19)

This register can be read from and written to. Refer to the Data Manual for details of each bit within this register.

***FUNCTION* GetChannelMuteData**

This function returns the Channel Mute Register value

```
Byte function GetChannelMuteData()
{
  SlaveAddr = 0x34;
  RegAddr = 0x19;
  BYTE HexData = ReadRegisterData(SlaveAddr, RegAddr);
}
```

***FUNCTION* SetChannelMuteData**

This function sets the individual channel mute.

```
function SetChannelMuteData ()
{
  SlaveAddr = 0x34;
  BYTE Data = 0x??; //Set data per table in Section 3 in Data Manual
                                    // 0xFF = no channels muted
  SendRegisterData(SlaveAddr, Data);
}
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265