

Programming I²S on the Stellaris Microcontroller

Application Note

Ryan Hunt

Design Team 6

November 13, 2011

Contents

Abstract	3
Keywords.....	3
Introduction and Background	3
Objective	4
Programming the Stellaris	5
Include Files	5
Define Variables	5
The Interrupt Handler	6
Main Program	7
Conclusion.....	8
References	8

Abstract

The I²S is a powerful interface than can digitally transmit stereo information. This application note discusses how to implement I²S reception from an external ADC on the Stellaris microcontroller.

Keywords

Inter-Integrated Sound, I²S, Stellaris, Microcontroller

Introduction and Background

Inter-Integrated Sound, also known as Inter-IC Sound, Integrated-Interchip Sound, and I²S, is a mode of transmitting digital audio information. It can be configured to transmit either stereo or mono sound. In this design project, I²S is used to transmit audio samples from the PCM1808 ADC to the Stellaris LM3S9B90 microcontroller.

The I²S bus has three lines: the continuous serial clock (SCK), the word select (WS), and the serial data line (SD). The SCK is the bit clock that drives both the transmitter and the receiver. The WS clock determines if the right or left audio signal is currently being transmitted for stereo signals. When WS is zero, channel 1 (left) is transmitted. When WS is one, channel 2 (right) is transmitted. The SD line contains the actual data being transmitted. SD transmits the MSB first. Either the transmitter or the receiver can be configured as the master, or an external controller can be the master. These system configurations and bit timing can be seen in Figure 1.

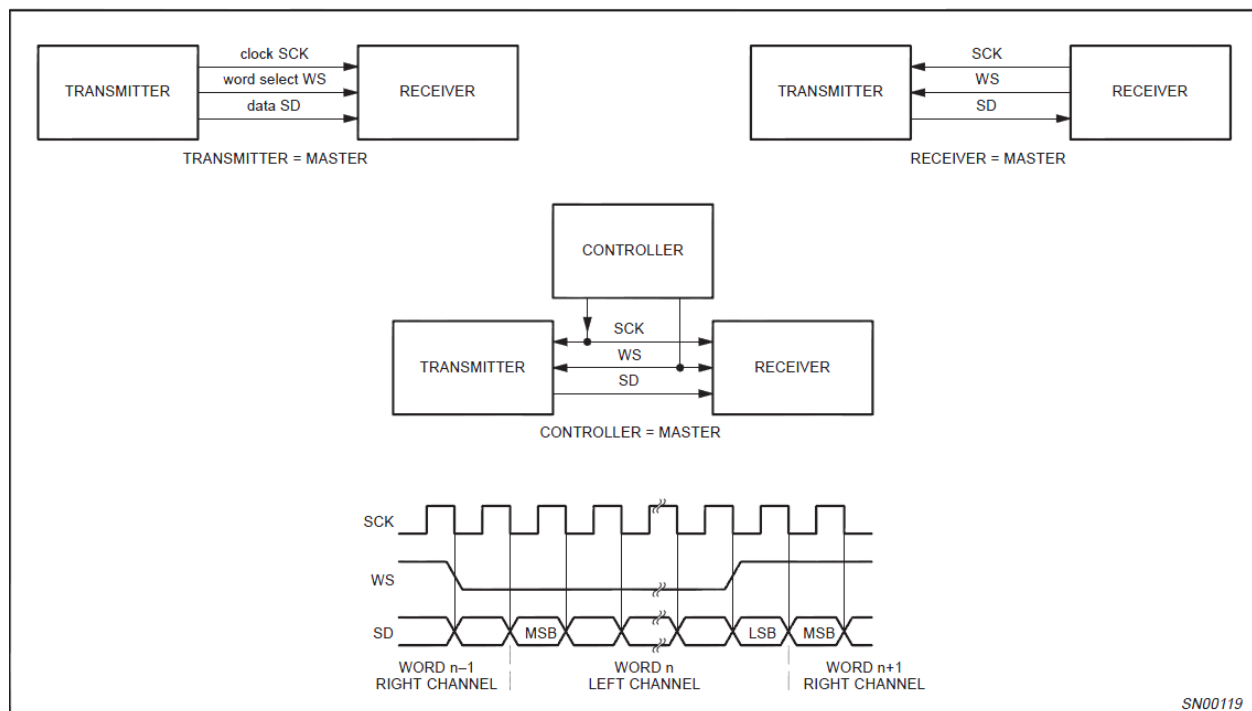


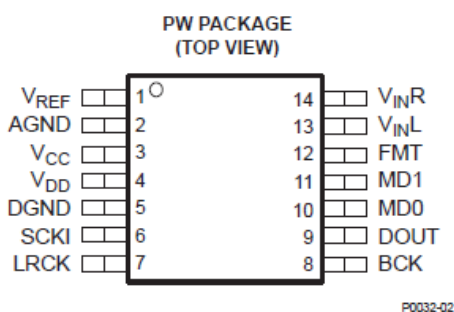
Figure 1 - I²S bus

Objective

The objective of this document is to demonstrate how to program the I2S on the Stellaris, including setting the GPIO pins, initializing the I2S module, registering and programming the interrupt handler, and storing the data. Note that programming requires the StellarisWare module library found on the TI websites at <http://www.ti.com/tool/sw-lm3s>.

Configuring the ADC

The ADC must be configured as the master module in this application. Figure 2 shows the pin assignments of the PCM 1808. To configure it as a master, the MD0 and MD1 pins must be driven high (3.3V). This configures the master mode of $256f_s$, as shown by the table in Figure 3. In addition, the FMT pin must be driven low, to send the I2S data in non-justified format, as shown by the table in Figure 4.



TERMINAL FUNCTIONS

TERMINAL NAME	PIN	I/O	DESCRIPTION
AGND	2	–	Analog GND
BCK	8	I/O	Audio data bit clock input/output ⁽¹⁾
DGND	5	–	Digital GND
DOUT	9	O	Audio data digital output
FMT	12	I	Audio interface format select ⁽²⁾
LRCK	7	I/O	Audio data latch enable input/output ⁽¹⁾
MD0	10	I	Audio interface mode select 0 ⁽²⁾
MD1	11	I	Audio interface mode select 1 ⁽²⁾
SCKI	6	I	System clock input; $256 f_s$, $384 f_s$ or $512 f_s$ ⁽³⁾
VCC	3	–	Analog power supply, 5-V
VDD	4	–	Digital power supply, 3.3-V
VINL	13	I	Analog input, L-channel
VINR	14	I	Analog input, R-channel
VREF	1	–	Reference voltage decoupling ($= 0.5 V_{CC}$)

(1) Schmitt-trigger input with internal pulldown (50-k Ω , typical)

(2) Schmitt-trigger input with internal pulldown (50-k Ω , typical), 5-V tolerant

(3) Schmitt-trigger input, 5-V tolerant

Figure 2 - PCM 1808 Pin Assignments

MD1 (Pin 11)	MD0 (Pin 10)	INTERFACE MODE
Low	Low	Slave mode (256 f _S , 384 f _S , 512 f _S autodetection)
Low	High	Master mode (512 f _S)
High	Low	Master mode (384 f _S)
High	High	Master mode (256 f _S)

Figure 3 - Interface Modes

FORMAT NO.	FMT (Pin 12)	FORMAT
0	Low	I ² S, 24-bit
1	High	Left-justified, 24-bit

Figure 4 - Data Format

Programming the Stellaris

Include Files

The I2S programming requires several modules in order to function. The required modules are i2s.c, gpio.c, interrupt.c, set_pinout.c, and sysctl.c. These modules can all be found in the driverlib folder in the StellarisWare home directory.

```
#include <string.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/i2s.h"
#include "driverlib/rom.h"
#include "drivers/set_pinout.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
```

Define Variables

Variables must be defined for left and right channel arrays, a Boolean indicating which sample is currently being received, and markers for the current position in the array. These are all static variables because they are used in the interrupt handler, which does not take inputs.

```
#define AUDIO_BUFFER_SIZE 4096
static unsigned long Left_Buffer[AUDIO_BUFFER_SIZE];
static unsigned long Right_Buffer[AUDIO_BUFFER_SIZE];
static int Left_Sample = true;
static int LBS = 0;
static int RBS = 0;
```

The Interrupt Handler

Next, the interrupt handler is written. It will get the interrupt status, handle any errors, and record the data in the FIFO into the appropriate buffer.

```
void My_I2S_Int (void)
{
    int ulStatus;

    //
    // Get the interrupt status to see what the interrupt is.
    //

    ulStatus = I2SIntStatus(I2S0_BASE, true);

    //
    // Clear the pending interrupts.
    //

    I2SIntClear(I2S0_BASE, ulStatus);

    //
    // Determine if there was an error
    //
    if(ulStatus & I2S_INT_RXERR)
    {
        // handle the error
    }

    //
    // Handle the RX service request
    //

    if(ulStatus & I2S_INT_RXREQ)
    {
        //
        //Retrive the data and put it in the buffer
        //
        if (Left_Sample)
        {

            I2SRxDataGet(I2S0_BASE, &Left_Buffer[LBS]);
            LBS++;
            if(LBS == 4096)
            {
                LBS=0;
            }
            Left_Sample = false;
        }
        else
        {
            I2SRxDataGet(I2S0_BASE, &Right_Buffer[RBS]);
            RBS++;
            if(RBS == 4096)
            {
                RBS = 0;
            }
        }
    }
}
```

```

        Left_Sample = true;
    }

}
}

```

Main Program

The main program function is now written. First, the I2S peripheral is enabled.

```

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

SysCtlPeripheralEnable(SYSCTL_PERIPH_I2S0);
SysCtlPeripheralReset(SYSCTL_PERIPH_I2S0);

```

Next, the GPIO pins and pin mux's are set. Note that since the Stellaris is receiving, only the Rx pins are set.

```

GPIOPinConfigure(GPIO_PA2_I2S0RXSD);
GPIOPinConfigure(GPIO_PD1_I2S0RXWS);
GPIOPinConfigure(GPIO_PD0_I2S0RXSCK);
GPIOPinConfigure(GPIO_PA3_I2S0RXMCLK);

GPIOPinTypeI2S(GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_3);
GPIOPinTypeI2S(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1);

```

The Rx module is now configured. It is configured for non-justified I2S format, 16-bit dual stereo mode, 24-bit samples, and set as a slave to an external clock.

```

I2SRxConfigSet(I2S0_BASE, I2S_CONFIG_FORMAT_I2S |
                I2S_CONFIG_MODE_DUAL |
                I2S_CONFIG_CLK_SLAVE |
                I2S_CONFIG_SAMPLE_SIZE_24 |
                I2S_CONFIG_WIRE_SIZE_32);

```

The FIFO buffer limit before triggering a FIFO service request is set to 4, and the master clock is set to be driven from an external oscillator.

```

I2SRxFIFOLimitSet(I2S0_BASE, 4);

I2SMasterClockSelect(I2S0_BASE, I2S_RX_MCLK_EXT);

```

Next, interrupts are configured and enabled. Pending interrupts are cleared, the interrupt handler is registered with the I2S module, and all interrupts are enabled.

```

I2SIntClear(I2S0_BASE, I2S_INT_RXERR | I2S_INT_RXREQ );

//
//Register my interrupt handler with the I2S controller
//

I2SIntRegister(I2S0_BASE, My_I2S_Int);

```

```
//
// Enable the interrupts for error and service request. Also,
// because the interrupt vector was allocated at compile-time, the
// peripheral interrupt must be enabled on the master controller.
//

I2SIntEnable(I2S0_BASE, I2S_INT_RXERR | I2S_INT_RXREQ);
IntEnable(INT_I2S0);
```

Finally, the I2S Receive module is enabled, and the program is placed into an infinite while loop, as all further handling is done at the interrupt level.

```
I2SRxEnable(I2S0_BASE);

while(1)
{
}
```

Conclusion

As demonstrated by this program, small I2S applications can be handled by a simple interrupt if they do not require time-intensive calculations. For example, the above program is not recommended with a PWM controller or other time-sensitive application; it is recommended that more programming be used along μ DMA to increase the speed of the I2S application.

References

- “I2S Bus Specifications” Philips Semiconductors. June 5, 1996.
- “Stellaris LM3S9B90 Microcontroller Data Sheet”. DS-LM3S9B90-9970. Texas Instruments Incorporated, 2007-2011
- “Stellaris Peripheral Driver Library User’s Guide”. SW-DRL-UG-8264. Texas Instruments Incorporated, 2007-2011.
- “SINGLE-ENDED, ANALOG-INPUT 24-BIT, 96-kHz STEREO A/D CONVERTER”. SLES177A. Rev August 2006. Texas Instruments Incorporated, 2007-2011.