EE 451 – LAB 3

Interrupts and Visualization Tools

In this laboratory you will use hardware and software interrupts in the C6713. You will also use visualization tools available in the CCS.

Introduction

In previous labs you have been using the polling method to generate a signal with the C6713. In polling mode, a function polls, or tests, the Transmit Ready bit (*XRDY*) of the MCBSP serial port control register (SPCR), until this indicates that the codec is ready to receive a new output sample. A new output sample is sent to the codec using the function *MCBSP_write*( ). Although polling is simpler than the interrupt technique, it is less efficient since the processor spends nearly all of its time repeatedly testing whether the codec is ready to transmit data. In the interrupt mode, an interrupt stops the current CPU process to that it can perform a required task initiated by an interrupt, and it is redirected to an interrupt service routine (*ISR*).

The *DSP/BIOS* real-time operating system available on the CCS provides real-time scheduling, analysis, and data transfer capabilities for an application running on the DSP. The DSP/BIOS has a preemptive real-time scheduler that determines which one of a number of different threads is executed by the DSP at any given time. Threads are DSP/BIOS objects that contain program code (functions). There are five different threads that can be used in a DSP/BIOS application:

- Hardware interrupts (*HWIs*) have the highest priority. Their execution is triggered by interrupts from peripherals and they always run to completion.
- Software interrupts (*SWIs*) are triggered from within a program. They run to completion unless preempted by a high priority SWI or HWI.
- Periodic functions (*PRDs*) are a special type of SWI triggered by a dedicated hardware timer.
- Tasks (*TSKs*) are created dynamically within a DSP/BIOS application, and they will start execution at the start of the DSP/BIOS application.
- Idle functions (*IDLs*) are executed repeatedly as a part of the lowest priority thread. They contain functions that communicate real-time data analysis from the DSP to the host.

The C I/O functions make it possible to access the host's operating system to perform I/O. The capability to perform I/O on the host gives you more options when debugging and testing code. However, calls to the C function *printf()* are computationally too expensive to be used within a real-time program. A LOG object inserted into a DSP/BIOS application sets up a buffer to which the function *LOG_printf()* can be used to append messages. The buffer contents are sent to a host computer in real time as part of the idle loop.

The Lab

Create a program that uses an ISR to read a signal from LINE IN and output it through the LINE OUT in real-time. Connect the function generator to the LINE IN input and verify that the program can reproduce the signal at the LINE OUT.

Part 1: Hardware Interrupts (HWIs)

1 Start CCS and begin a new project. You may use the same program template you used in the previous laboratory with some changes. You need to delete the statement *while(1).* There is no need to supply an explicit idle loop in a DSP/BIOS application. Modify the main() function to include the set up required to use interrupts, as depicted in Figure 1.

2 Create and add a configuration file to the project. Select File → New → DSP/BIOS Configuration. Select *dsk6713.cdb* as the template. Expand Scheduling and HWI – Hardware Interrupt Service Routine Manager and click on HWI_INT11. Right-click on HWI_INT11 and select properties to set the function to the ISR (in the sample template shown in Figure 1, it will be "_generate_sample"). Under the dispatcher tab, check Use Dispatcher. Note: there has to be an underscore preceding the name of your ISR.

3 Select Project → Build Options to the Compiler and Linker options. In the Advanced category of compiler options set the Memory Model option to Far Calls & Data.

4 Set the function generator to output a sinusoidal signal and connect it to the *LINE IN* of the board.

5 Connect an oscilloscope to the *LINE OUT* of the board and make sure the board is patching through the signal.

Part 2: Visualization Tools

1 Modify your code and use the printf() function to display the data from the function generator.

2 Turn on the CPU Load Graph from the DSP/BIOS → CPU Load Graph, run your code and record the CPU load. What is the CPU load?

3 Now we will create a log event. Add the following code to your program:

```
#include <log.h>
extern LOG_Obj LOG_YourLogVariable;
```

4 You need to print the value from the LINE IN input using a LOG_printf() function.

LOG_printf(&LOG_YourLogVariable,"output value %d\n", value_read_from_LINE IN);

5 Open the DSP/BIOS file and click on Instrumentation → LOG - Event Log Manager.Right-click on this item and Insert LOG. Rename the LOG name to YourLogVariable. Now in your code use LOG_printf() instead of the printf() function. Turn the message log on from DSP/BIOS → Message Log. Run your code and make sure it works.

6 What is the CPU Load?

```
//========= Lab3.c =========
//  This program patches through a signal from the LINE IN input
//

#include "dsk6713.h"
#include "dsk6713_aic23.h"                          // codec support
#include "dsk6713config.h"

Uint32 fs = DSK6713_AIC23_FREQ_48KHZ;               // set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE;        // select LINE IN input
void generate_sample(void)
{


  // code to read from LINE IN and write to LINE OUT

  return;

}


void main()
{

  // Set up needed to for interrupts

  IRQ_globalDisable();                               //disable interrupts

  DSK6713_init();                                    // call BSL to init DSK-EMIF,PLL)

  hAIC23_handle=DSK6713_AIC23_openCodec(0, &config);// handle(pointer) to codec
  DSK6713_AIC23_setFreq(hAIC23_handle, fs);          // set sample rate
  DSK6713_AIC23_rset(hAIC23_handle, 0x0004, inputsource);  // choose mic or line in
  MCBSP_config(DSK6713_AIC23_DATAHANDLE,&AIC23CfgData);// interface 32 bits to AIC23

  MCBSP_start(DSK6713_AIC23_DATAHANDLE, MCBSP_XMIT_START | MCBSP_RCV_START |
  MCBSP_SRGR_START | MCBSP_SRGR_FRAMESYNC, 220);     // start data channel

  CODECEventId=MCBSP_getXmtEventId(DSK6713_AIC23_codecdatahandle);//McBSP1 Xmit

  IRQ_map(CODECEventId, 11);                         //map McBSP1 Xmit to INT11
  IRQ_reset(CODECEventId);                           //reset codec INT 11
  IRQ_globalEnable();                                //globally enable interrupts
  IRQ_nmiEnable();                                   //enable NMI interrupt
  IRQ_enable(CODECEventId);                          //enable CODEC eventXmit INT11

  MCBSP_write(DSK6713_AIC23_DATAHANDLE,0);           //start McBSP interrupt outputting a sample

}
```

Figure 1. Program template for Lab3.