# Lab3
# Interrupts and Visualization Tools

Dr. Aly El-Osery

September 13, 2011

In this lab you will learn how to use hardware and software interrupts. Also, you will get familiar with the visualization tools and the real time data exchange capability of the C6713DSK.

## 1 Introduction

Interrupts are used to stop the current process in the CPU so that a certain task can be serviced. There are two different types of interrupts. The first kind is hardware interrupt, and the second is software interrupt. The problem with hardware interrupts, is that the CPU can't service multiple interrupts at the same time. This will cause events to be missed and interrupts to be ignored and real-time may be missed. Using software interrupts, different interrupts will be serviced based on the priority level which when designed correctly will help achieve real-time.

The C6713 comes with a variety of the visualization tools. Using the DSP/BIOS real-time analysis tools the performance of the DSP may be monitored, and hence provide a tool for optimization. Some of these tools include:

- CPU load graph.
- Execution graph.
- Message log.

In order to observe the operation of the DSP, the `printf` may be used to display the values of certain variables. This will work, but in the implementation of the `printf` function is not optimized and will load the CPU and eventually will prevent it from achieving real-time. Another way is to create a log-event and use the `LOG_printf` function instead.

## 2 Lab

### 2.1 Part 1: Hardware interrupts

1. What does "meeting real-time" mean ?.

2. Start a new project.
3. Create a DSP/BIOS file and add it to the project.
4. Create a hardware interrupt that is triggered by the data received from the MCBSP1. In the hardware service routine, read the data and retransmit it. Add to your source file `#include ''csl_irq.h''`. The CSL helps creating, configuring a nd using the interrupts.
   (a) Open the DSP/BIOS FILE and click on `Scheduling`.
   (b) Click on `HWI-Hardware Interrupt Service Routine Manager`.
   (c) Select `HWI_INT11` by right-clicking, then select `Properties`.
   (d) In the `interrupt source` field select `MCSP_1_Receive`.
   (e) In the `function field` enter the name of the hardware interrupt function that you want to create. *This field needs the assembly name of the function which is the name that you use in C preceded by an underscore.*
   (f) Click on the `Dispatcher` tab and select `Use Dispatcher`.
5. Add the following statements to enable interrupts after you initialize the board and the codec:

```
IRQ_map(IRQ_EVT_RINT1,11);      // Map McBSP1 using CSL function
IRQ_clear(IRQ_EVT_RINT1);
IRQ_globalEnable();             // Enable ints globally with bios function
IRQ_enable(IRQ_EVT_RINT1);      // Enable McBSP1 using CSL function
```

6. Test your hardware interrupt by connecting the input to either the function generator or to any other source and make sure that you are getting the same thing out.

## 2.2 Part 2: Software interrupts

Now we will try to create a software interrupt that will take advantage of the scheduling capability of the board. Here is how you can create a software interrupt.

1. Open the DSP/BIOS FILE and click on `Scheduling`.
2. Right-Click on `SWI-Software Interrupt Manager` and select `Insert SWI`.
3. Rename the SWI to something meaningful.
4. View the properties of the created SWI and insert the name of the function that you intend to create. *Again don't forget to precede the name of the software interrupt function by an underscore.*
5. In your source file create the software interrupt function that will be executed when an interrupt occurs.
6. You can `cut` and `paste` the same code that you had in your hardware interrupt function (HWI).
7. Now in your HWI interrupt function use the following function: `SWI_post(&swi_handle)`. Where `swi_handle` is the name of your software interrupt.
8. Test your code and make sure it works.

## 2.3 Part 3: CPU load

1. Turn on the `CPU Load Graph` from the `DSP/BIOS` menu, run your code and record the CPU load.
2. Add to your source file: `#include <stdio.h>`, and modify your program to use the `printf` function to display the data that you are reading.
3. What is the CPU load? Explain.
4. Now we will create a *log event*.
   (a) Open the DSP/BIOS FILE and click on `Instrumentation`.
   (b) Right-Click on `LOG - Event Manager` and select `Insert LOG`.
   (c) Rename the log event to something meaning full.
   (d) View the properties of the log event you just created and select `printf` to be the `datatype`.
   (e) Now, in your code, instead of using the `printf` use `LOG_printf`. *It is your responsibility to find out the required arguments for this function.*
   (f) Turn on the `Message Log` from the `DSP/BIOS` menu.
   (g) Run your code and make sure it works.
5. What is the CPU load?
6. Now try to find the CPU load difference between configuring the MCBSP to read 16bits from one channel and then do another 16bit read to get the other channel, and the configuring the MCBSP to read 32bit at once and then use logic and bit operations to separate the two channels.
   (a) set the sampling rate to 8kHz
   (b) monitor the CPU load while performing a 32-bit read from the codec.
   (c) monitor the CPU load with the default setting that allow you to read only 16 bits at a time.
   (d) Which approach is faster and why?