Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine

EE3T Study Project: Real-Time Digital Signal Processing with TMS320C6000

Laboratory 4 – Interrupt I/O

Objectives

- Learn how to use interrupt-driven I/O.

Setup

Map `\\reserver\sp_data` to `m:`. The files that are required for this lab can then be found in `m:\C6x\labs\lab4` and should be copied to an appropriate subdirectory in your workspace on `h:` for use. This should be the same directory in which you create the project below.

Exercise 1: Data sampling

In this first exercise you will use interrupts to sample an input waveform and output it again after half-wave rectification.

Creating the project and configuration files

1. Create a new project called `lab4.mak` and save it in an appropriate folder on `h:`.

2. The first step in configuring hardware interrupts is to setup the link between the physical interrupt and the ISR. We want the ISR to be triggered whenever a sample is received by the McBSP. Create a new DSP/BIOS Configuration file using the `dsk6711.cdb` template.

3. It is now necessary to associate a particular C function (which you are yet to write) with the desired hardware interrupt. Expand the HWI module in the configuration file. Examine HWI_INT4 through HWI_INT15, looking for the McBSP serial port receive interrupt `MCSP_0_Receive`. This is easiest if you click on HWI_INT4 (to show the properties field) and then cursor down the list.

4. Open the properties of the appropriate HWI_INT by right-clicking and selecting Properties. Enter the name of the ISR routine in the `function:` textbox. In our case the ISR will be a C routine, so its name must have a leading underscore, e.g. `_FunctionName`. Enter a function name of your choice (do *not* use `_ISR` since this is a reserved label). You will later write a C function with your chosen name that services the interrupt. Lastly, check the `Use Dispatcher` box so that the configuration tool will automatically take care of saving/restoring the context when an interrupt occurs.

5. Save the file as `lab4.cdb`. The configuration tool creates all the code required to configure the interrupt selections you made, and also creates the interrupt vector table for you.

6. Add the configuration file to the project, along with the linker command file `lab4cfg.cmd`.

Project options

1. Set the compiler options by copying the contents of `compiler.txt` to the `Project:Options:Compiler` textbox.
2. Set the assembler options to `-gls`.
3. Set the linker options by copying the contents of `linker.txt` to the linker window.

The C program shell

1. You are provided with a program `intio.c` whose listing is given in Appendix A. Add this file to the project.

This program does the following:

- Initializes the chip and board support libraries through calls to the pre-defined functions `CSL_Init` and `BSL_init`.
- Initializes the codec through the function `codec_init`, which in turn calls three BSL functions. First, it uses the function `AD535_open` to open the on-board AD535 codec and assign it to the handle `hAD535`. This handle is then reset through `AD535_reset`. Finally it is configured using `AD535_config` with the configuration structure `my_AD535_Config`.[1]
- Initializes hardware interrupts through the function `init_HWI`. This function first maps an event to a physical interrupt number through a call to the CSL function `IRQ_Map`, you should ensure that the interrupt number here matches the interrupt number you chose when setting up the configuration file. This particular interrupt is enabled through the CSL function `IRQ_Enable`. Finally, all interrupts are enabled through the DSP/BIOS API call `HWI_enable`. It is important that any initialization you require is performed in `main` before the call to `init_HWI`, since as soon as interrupts are globally enabled an interrupt could potentially occur.
- Reads a value from the codec. This is necessary since we must clear the McBSP receive register so that it can start generating interrupts when it receives a value.
- Enters an endless while loop. Effectively it is waiting here for interrupts to occur.

Interrupt service routine

You should now write a function that services the interrupt. This function should have the same name as the function name you assigned in the configuration file above. This function should perform the following operations:

1. Read in a sample from the codec. For this purpose you can use the special function `AD535_HWI_read` which is defined in the `bsl_extra.h` header file. This function is essentially the same as the BSL function `AD5353_read` except that it does not poll the AD535 to check that it has a sample ready to be read; since we are using this within an ISR that is only triggered when a sample is received we can safely assume that a sample is ready to be read in.

[1] There is another BSL call, `MCBSP_setfree(0)` which is essentially a bug fix used to reduce the chances of the codec locking up.

2. Half-wave rectify the sample.

3. Write out the rectified value. Again you can use the special function `AD535_HWI_write` that assumes the codec is ready to write a value.

Note that the output waveform will only look half-wave rectified if the input from the oscilloscope is below a certain frequency. Why is this?

Exercise 2: Interrupt-driven sine wave

Modify the program from Exercise 1 so that within the interrupt routine a sine wave is generated using the lookup-table method from Lab 3. As well as modifying the interrupt service routine, you will need to change the interrupt source so that the ISR is entered when the McBSP is ready to write a sample (rather than when it has received a new sample). This will require the following modifications:

1. Modify the configuration file so that the ISR is associated with the `MCSP_0_Transmit` interrupt.

2. Modify the function `init_HWI` to reflect the changes you made to the configuration file. Note that in the CSL the transmit interrupt is denoted as `IRQ_EVT_XINT0`.

3. In the main program you should replace the `AD535_read` call with an equivalent `AD535_write` call so that the McBSP will start generating transmit interrupts.

Appendix A

```
/*

INTIO.C    Interrupt I/O lab

DBW 25/4/01

*/

/***************************************
* Function prototypes
***************************************/
void codec_init(void);
void init_HWI(void);

/***************************************
* Include files
***************************************/
#include <c6x.h>       /* C6000 compiler definitions header */
#include <c6211dsk.h>  /* C6000 DSK definitions header */
#include <math.h>

#include <csl.h>  /* CSL headers */
#include <irq.h>
#include <mcbsp.h>

#include <bsl.h>  /* BSL headers */
#include <bsl_ad535.h>
#include <bsl_extra.h>

/***************************************
* Global variables
***************************************/
AD535_Handle hAD535;
AD535_Config my_AD535_Config = { AD535_LOOPBACK_DISABLE,
                          AD535_MICGAIN_OFF,
                          AD535_GAIN_0DB,
                          AD535_GAIN_0DB
                        };

/***************************************
* Main routine
***************************************/
void main(){

  /* Initialize the chip support library, required */
  CSL_Init();
  /* Initialize the board support library, required */
  BSL_init();
  /* open and configure the local codec */
  codec_init();
  /* initialize hardware interrupts */
  init_HWI();
```

```
  /* read a value to start generating interrupts */
  AD535_read(hAD535);

  /* loop indefinitely, waiting for interrupts */
  while(1) {};

}

/************************************
* Codec functions
************************************/
void codec_init()
{
  /* Use BSL routines to open , reset, and configure the codec */
  hAD535 = AD535_open(AD535_localId);
  AD535_reset(hAD535);
  AD535_config(hAD535, &my_AD535_Config);
  MCBSP_setfree(0);
}

/************************************
* Initialize hardware interrupts
************************************/
void init_HWI(void)
{
  IRQ_Map(IRQ_EVT_RINT0,11);  /* Map interrupt to McBSP0 */
  IRQ_Enable(IRQ_EVT_RINT0);  /* Enable McBSP0 interrupt */
  HWI_enable();               /* global enable HWI    */
}

/************************************
* INTERRUPT SERVICE ROUTINE
* Write the implementation of your ISR here
************************************/
```