DSP II: ELEC 4523

SWI Module

Objectives

- Become familiar with SWI module and its use

Reading

- SPRU423 TMS320 DSP/BIOS Users Guide: Software Interrupts (section)
- PowerPoint Slides from DSP/BIOS II Workshop: Module 5
- Code Composer Studio Online Help: SWI Module

Lab Module Prerequisites

It may be useful to complete the PRD module before this module.

Description

SWI objects are threads that run at a lower priority than hardware interrupts and higher than tasks. An SWI is triggered to run through function calls. SWI objects are used when the necessary processing has less critical time constraints or occurs at a slower rate than processing handled by HWIs. SWIs run to completion unless preempted by an HWI or another SWI of higher priority.

To create an SWI open the configuration file, right click on Scheduling->SWI and select Insert SWI. This will add a new SWI object. You can right click on the object and select Properties to change its properties. Put the function name in for the function you want to handle the SWI and set the SWI priority. Figure 1 shows a view of the configuration file where the SWIs can be seen in their corresponding priority. Notice that the `KNL_swi` is the lowest priority and it is reserved. This SWI is used to schedule the tasks.

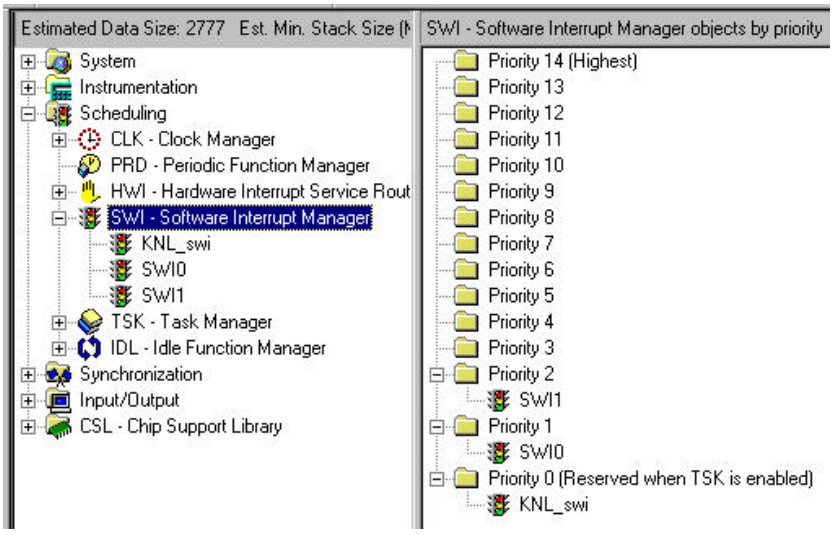

Figure 1: Configuration file showing SWIs and priorities

An SWI can be posted using the following functions

```
SWI_andn
SWI_dec
SWI_inc
SWI_or
SWI_post
```

When an SWI gets posted it gets put on the list of pending SWIs. If the currently running thread is of a lower priority the newly posted SWI gets executed and the running thread gets preempted. Once the SWI is finished execution returns to the preempted thread. If the currently running thread is of the same or higher priority then the posted SWI runs after all thread of higher or same priority are run. SWI threads cannot block or suspend execution to wait for a resource or event.

Each SWI has a mailbox (not to be confused with an MBX message). This mailbox is a 32-bit number which can be used either to determine whether to post the SWI or as values that can be used within the SWI function. The functions that post an SWI regardless of the mailbox value are

- `SWI_post` does not modify the value of the SWI object mailbox when it is used to post a software interrupt.
- `SWI_or` sets the bits in the mailbox determined by a mask that is passed as a parameter, and then posts the software interrupt.
- `SWI_inc` increases the SWI's mailbox value by one before posting the SWI object.

The functions that post an SWI when the value of the mailbox is zero are

- `SWI_andn` clears the bits in the mailbox determined by a mask passed as a parameter.
- `SWI_dec` decreases the value of the mailbox by one.

Differences between the functions are summarized in Table 1.

Table 1: SWI object function differences, taken from *SPRU423 TMS320 DSP/BIOS Users Guide*

Action	Treats Mailbox as Bitmask	Treats Mailbox as Counter	Does not Modify Mailbox
Always post	SWI_or	SWI_inc	SWI_post
Post if it becomes zero	SWI_andn	SWI_dec	—

When an SWI runs it can access its mailbox through the function call `SWI_getmbox`. The function `SWI_getmbox` returns the value of the mailbox right before the SWI was taken off the posted SWI list. The value of the mailbox is "latched" when the SWI is executed. When the SWI is executed the mailbox value is reset to its initial value. This value can be changed while the SWI is running. However, the value returned by `SWI_getmbox` is not affected.

An SWI only runs one time even though it may have been posted several times before it is run. If a section of code must be executed multiple times depending on the number of times the SWI was posted the function `SWI_inc` can be used. This will increment the mailbox of the SWI every time it is called. When the SWI executes it can get the mailbox value and run the code the number of times specified by the mailbox value.

Suppose an SWI needs to run only after two or more events occur. The function `SWI_andn` can be used to clear bits in the mailbox of an SWI. Each event clears a different bit and when all the events have occurred the SWI will be posted.

Suppose an SWI needs to run different code depending on the event that caused the SWI to be posted. The function `SWI_or` can be used to set bits in the mailbox of an SWI. Each event sets a different bit and when the SWI is posted it checks the bit that is set and runs different code depending on which bit is set.

If multiple occurrences of the same event must occur before and SWI is posted then the function `SWI_or` can be used to decrement the initial value of the SWI until it reaches zero. When it reaches zero the SWI will execute.

Laboratory

Part 1

- In this part of the lab you will create two SWI that post each other.
- Create a new project called `swilab`.
- Create a new DSP/BIOS Configuration file and use the `C6xxx.cdb` template for use with the simulator.
- Save the file as `swilab.cdb` and include it in your project. Also include the `swilabcfg.cmd` file.
- If using the simulator then change the RTDX interface to Simulator by right clicking on Input/Output->RTDX and bringing up the properties. Change the RTDX mode to Simulator. If you do not do this then when you load your program you will see the error "RTDX application does not match emulation protocol." If you are loading onto an EVM or DSK you shouldn't need to change this.
- Create a LOG object by right clicking on Instrumentation->LOG and selecting Insert LOG. Change the name to `trace`. Set its properties to have a length of 128 and be a fixed buffer.
- Create two SWIs with the following properties
  - Name: `SWI0`, priority: 1, mailbox: 0, function: `_funSWI0`.
  - Name: `SWI1`, priority: 1, mailbox: 0, function: `_funSWI1`.
- Create a main.c file and include a `main` function that does posts `SWI0` using `SWI_post`. Include this file in your project.
- In the main.c file make functions for your SWI, `funSWI0` and `funSWI1`. Each function should print to the `trace` LOG object once at the beginning of the function to indicate that the particular SWI is starting and once at the end of the function to indicate that the particular SWI is ending. In-between the prints each SWI should post the other SWI using `SWI_post`.
- The basic structure of main.c should be:

```
#include <std.h>
#include <swi.h>
#include <log.h>
#include "swilabcfg.h"

void main()
{
      SWI_post(&SWI0);
}

void funSWI0()
{
      /* code here */
}

void funSWI1()
{
      /* code here */
}
```

- Build and load your project.
- Open the log view, DSP/BIOS->Message Log.
- Run the program and record the results. You will need to halt the processor to allow the trace log to be updated. The reason for this is the CPU does not have any idle time to pass the data to the host for viewing.

Part 2

- Change the priority of `SWI1` to 2.
- Build and load your project.
- Run the program and record the results.

Part 3

- In this part of the lab you will have a periodic function that will post `SWI0`. `SWI1` will have a mailbox of 1 which will get decremented by `SWI0`. This will cause `SWI1` to run at half the rate as `SWI0`.
- Copy the `main.c` file from above to a new file `main3.c`. Remove the `main.c` from your project and add `main3.c`.
- Add a periodic object to the configuration file by right clicking on Scheduling->PRD and selecting Insert PRD. This will insert a periodic object called `PRD0`. Change the properties of `PRD0` so that the function it calls is `_funPRD0` and the period is 1.
- Change the configuration file so that SWI0 and SWI1 have the same priority and SWI1 has an initial mailbox value of 1.
- Change the `main3.c` file so that
  - function `funPRD0` posts `SWI0`
  - function `funSWI0` decrements the mailbox for `SWI1`
  - function `funSWI1` just prints out the same text as in part 1
  - function `main` does nothing
- Build and load your project.

- Run the program and record the results.

Part 4

- In this part of the lab you will have a periodic function that will post `SWI0`. `SWI1` will have a mailbox of 1 which will get decremented by `SWI0`. This will cause `SWI1` to run at half the rate as `SWI0`. Also, you will have another SWI, `SWI2`, that will get posted only when `SWI0` and `SWI1` run. They will use the function `SWI_andn` to clear bits in the mailbox of `SWI2`.
- Copy the `main3.c` file from above to a new file `main4.c`. Remove the `main3.c` from your project and add `main4.c`.
- Add a new SWI called `SWI2` to the configuration file. Set the properties so that it calls the function `funSWI2`, its mailbox is 3 and its priority is 1.
- Change the `main4.c` file so that
  - function `funSWI0` decrements the mailbox for `SWI1` and clears a bit in the `SW2` mailbox using `SWI_andn`
  - function `funSWI1` clears a different bit in the `SW2` mailbox using `SWI_andn`
  - function `funSWI2` just prints out text saying it is starting and ending
- Build and load your project.
- Run the program and record the results.

Part 5

- In this part of the lab you will have a periodic function that will post `SWI0`. `SWI1` will have a mailbox of 1 which will get decremented by `SWI0`. This will cause `SWI1` to run at half the rate as `SWI0`. Also, you will have another SWI, `SWI2`, that will get posted when either `SWI0` or `SWI1` run. They will use the function `SWI_or` to set bits in the mailbox of `SWI2`. When `SWI2` runs it will indicate which function posted it.
- Copy the `main4.c` file from above to a new file `main5.c`. Remove the `main4.c` from your project and add `main5.c`.
- Change `SWI2` in the configuration file so that its mailbox is 0 and its priority is 2.
- Change the `main5.c` file so that
  - function `funSWI0` decrements the mailbox for `SWI1` and sets a bit in the `SW2` mailbox using `SWI_or`
  - function `funSWI1` sets a different bit in the `SW2` mailbox using `SWI_or`
  - function `funSWI2` prints out text saying it is starting and ending and also a message indicating which bit was set in the mailbox
- Build and load your project.
- Run the program and record the results.