DSP II: ELEC 4523

Implicit Instrumentation and Kernel/Object View

Objectives

- Become familiar with the Execution Graph and the CPU Load Graph.

Reading

- SPRU423 TMS320 DSP/BIOS Users Guide: Implicit DSP/BIOS Instrumentation (section)
- Code Composer Studio Online Help: Execution Graph, If Too Few Events are Shown in the Execution Graph

Lab Module Prerequisites

It would be helpful to have completed the PRD module and the SWI module.

Description

Code needed to display the execution graph and the CPU load graph are built into every DSP/BIOS program. This code is very efficient and can be enabled or disabled as desired. The RTA Control Panel Analysis Tool allows turning on and off of different system logging. It is opened by selecting DSP/BIOS->RTA Control Panel.

Execution Graph

The execution graph is a tool that shows the processing and states of SWI, PRD, TSK, SEM and CLK objects. The execution graph does not time stamp every event but keeps track of CLK and PRD events for time reference. Since it doesn't keep track of time stamps the time scale is not linear. Ticks corresponding to CLK and PRD events give time reference. An example graph is shown in Figure 1. Notice that there is other informaiton being shown besides the times the objects are running. In the example there are two periodic threads, `PRD0` and `PRD1`. `PRD0` runs twice as often as `PRD1`.

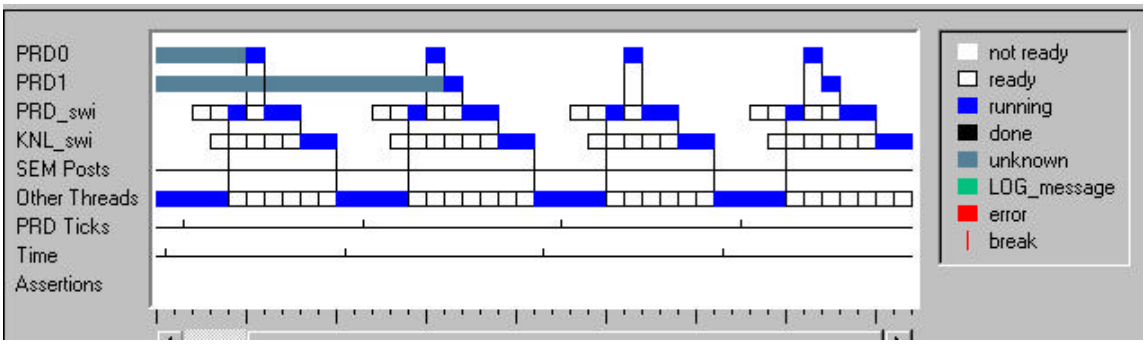


Figure 1: Execution graph example

The execution graph data is sent to the host through a LOG object called `LOG_system`. If you find that you want more information to be displayed between breaks there are a few things you can do:

- In the configuration file increase the buffer size of `LOG_system`
- Disable types of event logging that you are not concerned about in the RTA Control Panel
- Increase the polling rate of the RTA Control Panel by right clicking on the RTA Control Panel window and selecting Properties

CPU Load Graph

CPU load is the amount of time the DSP is doing work compared to the total time or the number of instruction cycles used for work compared to total time. The time the CPU is not working is considered idle time. With CCS 2.0 the CPU load graph does not work with the simulator.

Figure 2 shows and example CPU load graph. The plot shows a time line of CPU load. At the bottom is the peak CPU load as well as the last CPU load.

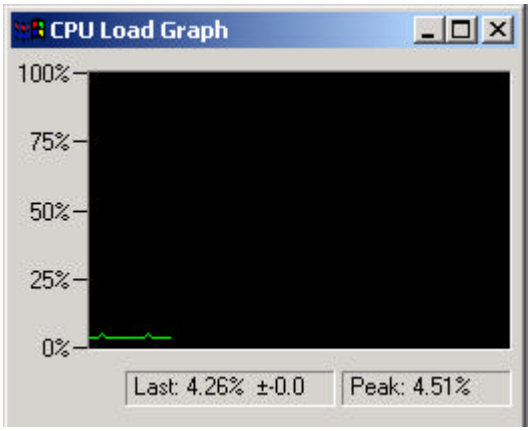


Figure 2: Example CPU load graph

Kernel/Object View

The Kernel/Object View gives information about the currently running objects on the target. There is information for the KNL, TSK, MBX, SEM, MEM and SWI objects. Most of the information in the display is self explanitory. It basically gives you the current state of each of the objects on the target. If you stop your code the display will update with the current object information. Changes in the display show up in red. An example is shown in Figure 3.

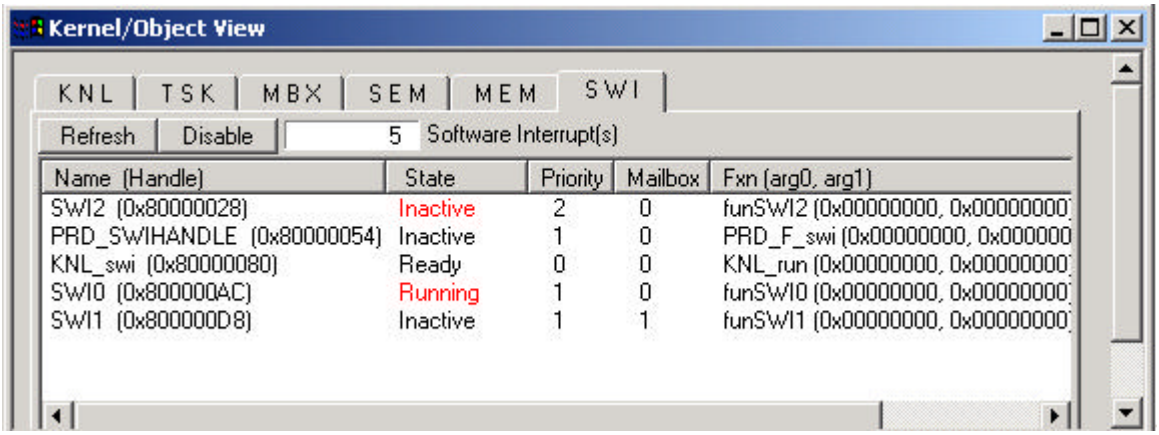
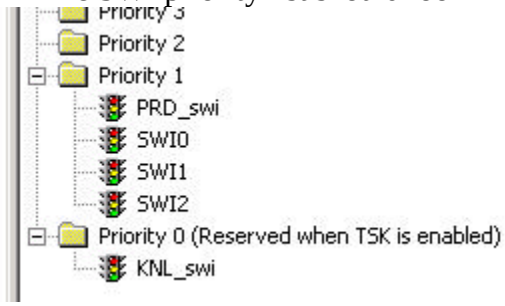


Figure 3: Kernel/Object View example

Laboratory

- Create a new project called `instlab`.
- Create a new DSP/BIOS Configuration file and use the `C6xxx.cdb` template.
- Save the file as `instlab.cdb` and add it in your project. Also add the `instlabcfg.cmd` file.
- In order to see the CPU load graph the project must be loaded onto a DSP device like the ʼC6701 EVM or the ʼC6711 DSK. If you are loading onto a board then leave the RTDX interace on JTAG. 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.
- Change the `LOG_system` object buffer size to 1024.
- Create the following objects with corresponding properties:
  - PRD object, name: `PRD0`, period: 1, function: `_funPRD0`.
  - SWI object, name: `SWI0`, priority: 1, mailbox: 0, function: `_funSWI0`.
  - SWI object, name: `SWI1`, priority: 1, mailbox: 2, function: `_funSWI1`.
  - SWI object, name: `SWI2`, priority: 1, mailbox: 0, function: `_funSWI2`.
- The SWI priority list should look like



- Create a main.c file and include the following code.

```
#include <std.h>
#include <swi.h>
#include <prd.h>
#include "instlabcfg.h"

void main()
{
}

void funPRD0()
{
      SWI_post(&SWI0);
}

void funSWI0()
{
      SWI_dec(&SWI1);
```

```
      SWI_or(&SWI2,1);
}

void funSWI1()
{
      SWI_or(&SWI2,2);
}

void funSWI2()
{
}
```

- Build and load your project.
- Open the execution graph, CPU load graph and Kernel/Object View.
- Put breakpoints in the four objects.
- Run the program and record the results. Show printouts of the graphs. Step through the code and record how the object information changes as the SWI functions are called.

Part 2

- Change the SWI2 priority to 2.
- Build and load your project.
- Run the program and record the results. Show printouts of the graphs. Step through the code and record how the object information changes as the SWI functions are called.