DSP II: ELEC 4523

PIP Module

Objectives

- Become familiar with pipes and their use with SWI objects

Reading

- SPRU423 TMS320 DSP/BIOS Users Guide: Data Pipe Manager (PIP Module) (section)
- PowerPoint Slides from DSP/BIOS II Workshop: Module 6
- Code Composer Studio Online Help: PIP Module

Lab Module Prerequisites

Be sure to do the SWI module before the PIP module.

Description

Pipes are objects that maintain a buffer which is broken up into fixed length frames. Frames can only be accessed one frame at a time. On the writing side a frame is allocated, filled with data and then put back in the pipe. On the receiving end the frame is received, processes and returned to the pipe. There should be only one reader and writer for a pipe. Frames are a fixed length but the program doesn't have to fill the whole frame.

To write data to a frame the following process should be followed:

- Check to see if there are empty frames available with `PIP_getWriterNumFrames`.
- Allocate an empty with the function `PIP_alloc`. This reserves a frame for writing.
  - If there are more empty frames still in the pipe the function will call the `notifyWriter` function before returning. This may seem a little strange since the writer may be the SWI that is doing the writing. However, remember that SWIs run to completion and that if an SWI gets posted while it is running it will run again when it is completed.
- Get the address of the frame and its size with `PIP_getWriterAddr` and `PIP_getWriterSize`.
- Write data to the frame using the address returned above. The address will point to the beginning of the frame where data can be written.
- Put the frame back in the pipe using `PIP_put`. If the whole frame was not filled up, the size of the frame can be set by calling `PIP_setWriterSize` before the call to `PIP_put`.
  - The call to `PIP_put` calls the `notifyReader` function.

The basic structure of the writer function is:

```
/* check to see if frames are available before accessing them */
if(PIP_getWriterNumFrames(&dataPIP))
{
```

```
        /* allocate a new frame of data */
        PIP_alloc(&dataPIP);
        /* get the address to the new frame */
        addr=PIP_getWriterAddr(&dataPIP);
        /* get the size of the frame in words */
        size=PIP_getWriterSize(&dataPIP);

        /* Put code to fill the frame here */

        /* put the new frame of data into the pipe */
        PIP_put(&dataPIP);
    }
    else {
        return;
        // LOG_error("no frames available",0);
    }
```

The process to read fames from a pipe is very similar to the process for writing frames to a pipe. To read data from a frame the following process should be followed:

- Check to see if there are full frames available with `PIP_getReaderNumFrames`.
- Get a full frame with the function `PIP_get`.
  - If there are more full frames still in the pipe the function will call the `notifyReader` function before returning. This may seem a little strange since the reader may be the SWI that is doing the reading. However, remember that SWIs run to completion and that if an SWI gets posted while it is running it will run again when it is completed.
- Get the address of the frame and its size with `PIP_getReaderAddr` and `PIP_getReaderSize`.
- Process the data in the frame using the address returned above. The address will point to the beginning of the frame.
- Put the frame back in the pipe using `PIP_free`.
  - The call to `PIP_free` calls the `notifyReader` function.

The basic structure of the reader function is:

```
    /* check to see if frames are available before accessing them */
    if(PIP_getReaderNumFrames(&dataPIP)>0)
    {
        /* allocate a new frame of data */
        PIP_get(&dataPIP);
        /* get the address to the new frame */
        addr=PIP_getReaderAddr(&dataPIP);
        /* get the size of the frame in words */
        size=PIP_getReaderSize(&dataPIP);

        /* Put code to use the frame here */

        /* free up the frame of data */
        PIP_free(&dataPIP);
    }
    else {
        return;
```

```
        // LOG_error("no frames available",0);
    }
```

When using the PIP module the `notifyWriter` and `notifyReader` functions can be used to synchronize the transfer of data. If the pipe is being used with a SWI then posts to the SWI can be done in these functions. Suppose there is a periodic function that generates data to be used by a SWI. It would be desirable to have the SWI be posted when the data is available. In the `notifyReader` function a call to `SWI_post` for that SWI can be used. When the periodic function puts the data in the pipe with `PIP_put` it will call the `notifyReader` function which will post the SWI.

Laboratory

Part 1

- In this part you will be creating two SWIs were one will generate some data and send it to the other SWI in a data pipe. The transmitting SWI will generate only 3 frames of data to send to the other SWI.
- Create a new project called `piplab`.
- Create a new DSP/BIOS Configuration file and use the `C6xxx.cdb` template for use with the simulator.
- Save the file as `piplab.cdb` and add it in your project. Also add the `piplabcfg.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 1024 and be a circular buffer.
- Change the `LOG_system` object to have a length of 1024.
- Create a PIP object with the following properties:
  - Framesize: 8
  - Number of Frames: 3
  - notifyWriter function: `_PIP0notifyWriter`
  - notifyReader function: `_PIP0notifyReader`
- Create two SWIs with the following properties
  - Name: `SWI0`, priority: 1, function: `_funSWI0`.
  - Name: `SWI1`, priority: 1, function: `_funSWI1`.
- One of the SWIs will be used to generate some data and the other will receive the data.
- Create a main.c file and include a `main` function that prints to `trace` a message saying the run is starting and posts the `SWI0`. Include this file in your project.
- In the main.c file make functions for your SWIs, `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. All other code in each SWI will go between these statements.

- In fun`SWI0`, which will generate the data, do the following
  - Make a `static` variable to hold the number of times the function has been called. Initialize it to zero.
  - Put in an `if` statement to check to see if the function has been called 3 times. If it has then return, if not then increment the loop counter. Before the return be sure to put a print to trace indicating that `SWI0` is ending.
  - Add code similar to the code in the Description section to get a frame of data from the pipe and put it back. Fill the frame with integers equal to the number in the frame times the loop number.
- In `funSWI1`, which will receive and print the data, add code similar to the code in the Description section to get a full frame of data from the pipe and put it back. Have the code print the contents of the frame to the `trace` LOG.
- Create a function `PIP0notifyWriter` and put a post to the `SWI0` in it.
- Create a function `PIP0notifyReader` and put a post to the `SWI1` in it.
- Build and load your project.
- Use the Execution Graph and the LOG manager to examine the processing.
- Run the program and record the results.
- Describe in detail the processing that is occurring.

Part 2

- Change the priority of `SWI1` to 2.
- Build and load your project.
- Run the program and record the results.
- How does the execution change? Describe in detail.

Part 3

- Change the priority of `SWI0` to 2 and the priority of `SWI1` to 1.
- Build and load your project.
- Run the program and record the results.
- How does the execution change? Describe in detail.