Porting a SPOX–KNL V2.2 Application to DSP/BIOS II

Robert Tivy and Arnie Reynoso *Software Development System/Semiconductor*

ABSTRACT

DSP/BIOS II supports a single set of programming APIs for both BIOS and SPOX–KNL that is as scalable as possible. In cases where similar APIs exist in both BIOS and SPOX, the BIOS APIs are used. This may present some incompatibilities between the current BIOS kernel APIs and their counterparts in SPOX v2.2. This document describes the major incompatibilities and provides instructions for porting a SPOX–KNL v2.2 application to DSP/BIOS II. For complete details, please refer to the DSP/BIOS User's Guide.

Contents

1 Overview

Incompatibilities in the following areas exist between SPOX–KNL v2.2 APIs and DSP/BIOS II APIs:

- System startup
- Stacks
- Software Interrupts (SWI module)
- Hardware ISRs (HWI module)
- Memory (MEM module)
- Streaming I/O and User Defined Device Drivers (SIO and DEV modules)
- Host I/O
- Configuration Tool (CDB file) configuration and conversion

The incompatibilities and instructions for porting a SPOX–KNL v2.2 application to DSP/BIOS II are described in the following sections.

2 System Startup

The major difference in the system startup is the way in which the function **main()** is called.

2.1 SPOX v2.2

In SPOX v2.2, **main()** is simply a task in the system like any other task and is statically created in the configuration file. It is special because:

- it is the first task that is created,
- it is created with the maximum task priority, and
- it will be the first task to run regardless of the priorities of other user tasks in the system.

The following pseudo-code illustrates the complete startup sequence:

```
boot.s62 calls startup() (in startup.c)
        startup() {
          SPOX_init(SYS);    /* initialize SYS backplane */
          KNL_exit();      /* schedule first SPOX task */
        }
        ...
        ...            /* spox() is the function
                       * entry point of the first
                       * Spox task
                       */
       spox() {          /* (in startup.c) */
         SPOX_start()      /* start all SPOX modules */
         SYS_start();      /* initialize APP backplane */
         main(argc, argv, envp);
    }
         ...
         ...
       main(argc, argv, envp) { /* user's main() */
         ...
         ...
       }
```

2.2 DSP/BIOS II

In DSP/BIOS II, **main()** becomes a special function, not a SPOX task. The DSP/BIOS II system doesn't start scheduling threads until the function **main()** returns; therefore, only a limited set of the DSP/BIOS II APIs can be called from the **main()**.

In direct contrast to the SPOX-KNL v2.2 processing, while **main()** is running and the DSP/BIOS II system not yet started,

- hardware interrupts are globally disabled
- the SWI and TSK schedulers are disabled

The recommended way to port an SPOX-KNL v2.2 application that contain a **main()** function that creates tasks, semaphores, etc., and possibly implements application run-time functionality is to rename **main()** to another name (for example, **smain()**). Then, using the GUI-based DSP/BIOS II configuration tool, you should statically create a **TSK** task with a maximum priority of 15 whose function is **smain()**, and also create an empty **main()** function in your C source file.

The following pseudo-code illustrates the complete startup sequence:

```
boot.s62
       BIOS_init();
       main(argc, argv, envp)
       BIOS_start()
```

Until **BIOS_start()** is called, execution threads will not run.

2.3 DSP/BIOS II Functions that Cannot be Called from Function main()

No blocking functions, and only single-threaded functions, may be called by **main()**. The following blocking functions cannot be called by **main()**:

- SEM_pend()
- MBX_pend() and MBX_post()
- SIO_get() and SIO_put()
- SIO_select()
- TSK_sleep()

2.4 Hooking into the Initialization Backplane

DSP/BIOS II allows users to insert their own functions into the initialization backplane using the GUI-based configuration tool. Previously, this was accomplished in SPOX by using the text–based configuration tool syntax.

Using the configuration tool, check the box labeled, “Call user init function” under the “Global Settings Properties.” Then enter the name of the user initialization function. This function will be called by the **GBL_init** macro, which is the first macro executed in the **BIOS_init()** function. In addition to calling the user function, **GBL_init** performs chip-level initialization, such as setting up various possible caches and initializing system buffers.

3 System Stack

On the C6x, DSP/BIOS II contains a system stack that is used while handling ISRs. This relieves the **TSK** task stacks from having to support ISR stack usage, allowing task stacks to be smaller. However, in addition to the space needed by the task’s function call dynamics, the task stack needs to contain enough extra space for a single preemption context. A preemption context is the CPU state that gets saved when a task is interrupted and preempted by a higher priority task. The CPU state includes all the general purpose registers in addition to certain special purpose registers and system variables.

4 Software Interrupts (SWI)

SWIs in DSP/BIOS II are traditional BIOS SWIs (previously called SIG in BIOS). They therefore conform to the BIOS priority-based scheduling model. SPOX SWIs were scheduled in FIFO ordering.

The run-time APIs of SPOX SWIs have been superceded by similar APIs in DSP/BIOS II. These new APIs consist of all the APIs that were present in BIOS, plus some additional ones to support dynamic creation and deletion. These additional APIs, **SWI_create()** and **SWI_delete()**, were available in SPOX but not BIOS, and their calling model (function parameters) have changed to more closely reflect the DSP/BIOS II environment.

The method in which the kernel calls DSP/BIOS II software interrupts now conforms to the traditional BIOS model. This has little impact on the actual SWI function itself, but there is one area of possible concern: when and how SWIs are called. In SPOX, SWIs were called from inside the kernel scheduler, **KNL_exit()**. They are now handled by the BIOS scheduler (**SWI_exec()**). In fact, the kernel scheduler **KNL_exit()** is itself an SWI.

In SPOX, the kernel scheduler calls an SWI once per separate posting. If an SWI is posted more than once, the kernel scheduler removes the SWI from a list, calls the SWI function, and if there are further postings to process, the kernel enqueues the SWI on the tail of the SWI list. It is then processed again after all other SWIs that are in front of this SWI in the list have been processed.

The BIOS scheduler only calls the SWI once, even if it has been posted multiple times before running. The following excerpt from the DSP/BIOS II documentation (**SWI_inc** reference page) explains how to simulate this SPOX calling model:

> If a software interrupt is posted several times before it has a chance to begin executing, – because HWIs and higher priority software interrupts are running – the software interrupt only runs one time. If this situation occurs, you can use **SWI_inc** to post the software interrupt. Within the software interrupt's function, you could then use **SWI_getmbox** to find out how many times this software interrupt has been posted since the last time it was executed.

To process the multiple postings, the SWI can either loop again for each posting, or post itself again with **SWI_post** (*not* **SWI_inc**), depending on the processing order desired. When an SWI posts itself, it is placed on the tail of its priority queue, thereby allowing other SWIs of equal priority an opportunity to run before running again. By reposting itself, an SWI is called in the same manner as SPOX v2.2 SWIs when multiple postings are pending.

Note: TSK_setpri() can no longer be called by SWIs. It can only be called from TSK level.

5 Hardware Interrupts (HWI)

The SPOX v2.2 ISR assembly macros (**C62_enter/C62_exit**) have been replaced with the generic ISR interface macros **HWI_enter/HWI_exit**. The parameters to the HWI macros are identical to the SPOX–KNL C62 macros, retaining the C62 prefix.

Aside from the name change, the main difference is in the **HWI_exit** macro. In SPOX, the **C62_exit** macro would potentially call the SPOX **KNL** scheduler, whereas in DSP/BIOS, the **SWI** scheduler is potentially called. The **KNL** scheduler is then called by the **SWI** scheduler.

ISRs in both SPOX and BIOS shared a common model. For the C6x, the main difference between ISRs in SPOX/BIOS and DSP/BIOS II is the addition of an **HWI** dispatcher. A given user ISR can now be configured to be handled (dispatched) by the **HWI** dispatcher. The user still has the option of handling the ISR in the traditional SPOX/BIOS method, but significant code savings can be achieved when the dispatcher is used. The C54x does not contain the dispatcher, so the traditional methods of handling ISRs still apply.

Two **HWI** objects are preconfigured by DSP/BIOS II, one used by the **CLK** module and another used by the RTDX module (which is hidden from the configuration tool). Both of these are hard-configured to use the **HWI** dispatcher, but the **CLK** module's dispatcher parameters (interrupt enable mask and cache control mask) are configurable by the user.

The interrupt enabling/disabling macros from SPOX v2.2 have been renamed to more closely match their counterparts in BIOS. The following translations apply:

Table 1. SPOX v2.2 Enabling/Disabling Macros

SPOX v2.2	DSP/BIOS II
C62_enable	C62_enableIER
C62_disable	C62_disableIER
C62_enableGIE	HWI_restore
C62_disableGIE	HWI_disable

There is no backward compatibility for the SPOX C62 APIs, so it is necessary to modify those source files that contain them.

6 Memory

The **MEM** module has been enhanced to provide finer granularity of control over static memory section configuration and usage. Please refer to the DSP/BIOS User's Guide and the DSP/BIOS II Configuration Tool Online Help for further information.

7 Streaming I/O and User-Defined Device Drivers (SIO and DEV)

Beginning with SPOX-KNL v2.1, the **SIO_ISSUERECLAIM** model was added to the SIO interface. With this change, two new functions were added to the DEV_Fxns table: **DXX_issue** and **DXX_reclaim**.

In DSP/BIOS II, **SIO_get()** and **SIO_put()** now use the device driver's issue and reclaim functions, making the DEV device driver interface more efficient and easier to write. The device drivers' input and output functions were removed making device drivers smaller. All device drivers are affected by this change, however updating the assorted device drivers is not difficult.

7.1 Required Device Driver Changes

- The functions **DEV_Fxn.input** and **DEV_Fxn.output** have been removed. All device drivers' input and output functions should be removed.
- The **DEV_Fxns** table has been alphabetized. The user must alphabetize the functions of the device drivers (and delete input and output).
- **DEV_Obj.model** has been removed. The 'device—>model' field has been moved to the **SIO_Obj** structure. All device drivers must be modified to remove any references to the **DEV_Obj.model**. Device drivers should behave as if they were opened in the **SIO_ISSUERECLAIM** model. All I/O should be done with calls to **DXX_issue()** and **DXX_reclaim()**.
- The timeout parameter to **SIO_reclaim()** has been deleted, and a **DEV_Obj.timeout** field has been added. This field is initialized by **SIO_create** (or the static **SIO** create process) to contain the timeout to wait for **SIO_reclaim().**
- Device drivers are now initialized by **DEV_init()** *before* **main(). SIO_startup()** initializes the static streams *after* **main().**

7.2 Streaming I/O Interface Changes

The two changes to the SIO interface are:

- A device-specific timeout is now specified when the stream is created (see above). **SIO_reclaim()**'s timeout parameter has been removed.
- **SIO_staticbuf()** API has been added. This is used to get statically created buffers from the stream when the SIO stream is configured using the gconf Configuration Tool.

8 HOST I/O (HST)

In SPOX, host I/O was achieved using the File I/O (**FIO** module). An **FIO** object was configured using the configuration tool and subsequently used to retrieve a Streaming I/O **(SIO** module**)** stream object and buffer. The **SIO** object and buffer were then used in the traditional method. In DSP/BIOS II, **FIO** has been removed and equivalent functionality can be achieved using an **HST** object, **DHL** device instance, and the **SIO** module. Alternatively, host I/O can be achieved using an **HST** object in conjunction with the **PIP** module. Please refer to the DSP/BIOS User's Guide for further details.

9 Configuration Tool (CDB file) configuration and conversion

SPOX-KNL v2.2 **CDB** files are not compatible with DSP/BIOS II, and there is no automated method for converting SPOX-KNL v2.2 **CDB** files to DSP/BIOS II **CDB** files. The user must manually recreate the CDB from from a DSP/BIOS II template.

9.1 Manually Recreating CBD Files

Code Composer Studio allows multiple configuration (CDB) files to be opened simultaneously. This will allow the user to easily copy objects from a SPOX v2.2 CDB file to a DSP/BIOS II CDB file. A step by step method to perform this task is described below:

- Use Code Composer Studio to open the SPOX-KNL configuration file (*.cdb). Select File→Open. In the Open dialog box, select the configuration file and click Open. Select NO when prompted to update the file.
- Create a DSP/BIOS configuration file within Code Composer Studio. Select File→New→DSP/BIOS Configuration. In the New dialog box, select the appropriate configuration template for your DSP chip and board and click OK. At his point, two configuration files are opened within Code Composer Studio.
- The user must manually recreate the following objects module in the new DSP/BIOS II CDB file:
  MEM, TSK, HWI, SIO and DGN (Note: SIOs cannot be created until device drivers are defined).
- The remaining user created objects (CLK, IDL, LCK, LOG, MBX, QUE, SEM, STS, DPI, and User-Defined Devices) from the SPOX_KNL CDB file to the DSP/BIOS II CDB file can be transferred using the cut (Ctrl+C) and paste (Ctrl+V) features as follows:
  - Highlight the object to be copied and press Ctrl+C to copy the selected object.
  - Move to the new configuration file and select the associated object manager and press Ctrl+V to paste the object into the new configuration file.

Repeat the above two steps for all remaining objects to be transferred.

Software Interrupts (SWI) and Host File Manager (FIO) have changed from SPOX_KNL v2.2 to DSP/BIOS II. See pertinent sections in this document for further information.

Additional functionality/flexibility has been added to the following object modules: Global setting (GBL), Memory Section Manager (MEM), Hardware Interrupts (HWI).

Be sure that the appropriate selections are made in the new configuration file. Refer to the DSP/BIOS User's Guide for further details.

10 References

1. TMS320C5400 DSP/BIOS User's Guide (SPRU326)
2. TMS320C6000 DSP/BIOS User's Guide (SPRS303)
3. Code Composer Studio (v1.2) DSP/BIOS online help

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.