Technical notes on using Analog Devices DSPs, processors and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

Expert Pin Multiplexing Plug-in for Blackfin® Processors

Contributed by Jagadeesh Rayala and Anand Kopparthi *Rev 2 – January 7, 2010*

Introduction

This EE-Note explains how to use the Expert Pin Multiplexing plug-in for VisualDSP++® development tools (release 5.0 or higher) to configure the Port registers in ADSP-BF51x, ADSP-BF52x, ADSP-BF534/BF536/BF537 (hereafter referred to as ADSP-BF537 processors), and ADSP-BF54x Blackfin® processors. The Expert Pin Multiplexing plug-in simplifies the task of generating the C and/or assembly code that is used to program the Port registers.

Pin Multiplexing

Blackfin processors feature a rich set of peripherals, which through a powerful pin multiplexing scheme, provides great flexibility to the external application space. The ADSP-BF51x processors group the many peripheral signals into three ports – Port F, Port G and Port H. For ADSP-BF52x and ADSP-BF537 processors, peripheral signals are grouped into four ports – Port F, Port G, Port H, and Port J. For ADSP-BF54x processors, peripheral signals are grouped into ten ports – referred to as Port A through Port J. Most of the associated pins are shared by multiple signals. The ports function as multiplexer controls. Every pin in Port A through Port J (Port F, Port G, and Port H for ADSP-BF51x, ADSP-BF52x and ADSP-BF537 processors) can also function as a GPIO pin. Any pin not used by a function can be configured in GPIO mode on an individual basis. By default, after reset, all pins are in GPIO mode. Neither GPIO output nor input drivers are active by default.

Every port has its own set of memory-mapped registers to control port multiplexing and GPIO functionality. Peripheral functionality must be explicitly enabled by the function enable registers (`PORTx_FER`, where x = F, G, or H for ADSP-BF51x, ADSP-BF52x, and ADSP-BF537 processors). The competing peripherals on Ports are controlled by the respective multiplexer control register (`PORTx_MUX` for ADSP-BF51x, ADSP-BF52x, and ADSP-BF54x processors and `PORT_MUX` for ADSP-BF537 processors). Any GPIO can be enabled individually and overrides the peripheral function if the respective bit in the `PORTx_FER` is cleared. To drive the pin in GPIO output mode, the respective direction bit must be set in the `PORTxIO_DIR` register (`PORTx_DIR_SET` for ADSP-BF54x processors). To make the pin a digital input, the input driver must be enabled in the `PORTxIO_INEN` register (`PORTx_INEN` for ADSP-BF54x processors). By default, all peripheral pins are configured as inputs after reset. However, GPIO input drivers are disabled to minimize power consumption and any need of external pulling of resistors on unused or don't care pins. For additional information on pin multiplexing, refer to the processor's *Hardware Reference* [1] [2] [3] [4].

Peripheral and GPIO configuration requires an in-depth understanding of the Port registers, bit field positions corresponding to different signals in all the registers, number of bits allocated for each bit field in all the registers, and the values that correspond to different signals in all the registers.

The Expert Pin Multiplexing plug-in provides an easy method of generating the code necessary to configure the Port registers. The Expert Pin Multiplexing tool allows you to generate the code without having to worry about internal details.

Installing the Expert Pin Multiplexing Plug-In

To install the Expert Pin Multiplexing plug-in in the VisualDSP++ 5.0 environment:

1. Extract the file `ExpertPinMux.dll` from the associated `.ZIP` file (`EE341v01.zip`) and place it in the VisualDSP++ `System` directory. If VisualDSP++ is installed on the C drive (default installation path), copy the attached file into the following directory:

   ```
   C:\Program Files\Analog Devices\VisualDSP 5.0\System
   ```

2. Register the `ExpertPinMux.dll` file by typing the following command line:

   ```
   C:\Windows\system32\regsvr32.exe ExpertPinMux.dll
   ```

   Note: Run `regsvr32.exe` from the `<install_path>\System` directory, not from the root directory.

Windows Vista users should run the command prompt as administrator in order to register the plug-in. The Expert Pin Multiplexing tool now appears on the `Plugins` page of the `Preferences` dialog box (`Settings` -> `Preferences`). The Expert Pin Multiplexing utility can be accessed from the `Tools` menu.

This plug-in is enabled only for ADSP-BF537, ADSP-BF54x, ADSP-BF52x, and ADSP-BF51x sessions of VisualDSP++ release 5.0 or higher.

Figure 1 shows the default state of the Expert Pin Multiplexing window. By default, the ADSP-BF512 processor is selected and all the list boxes are populated accordingly.

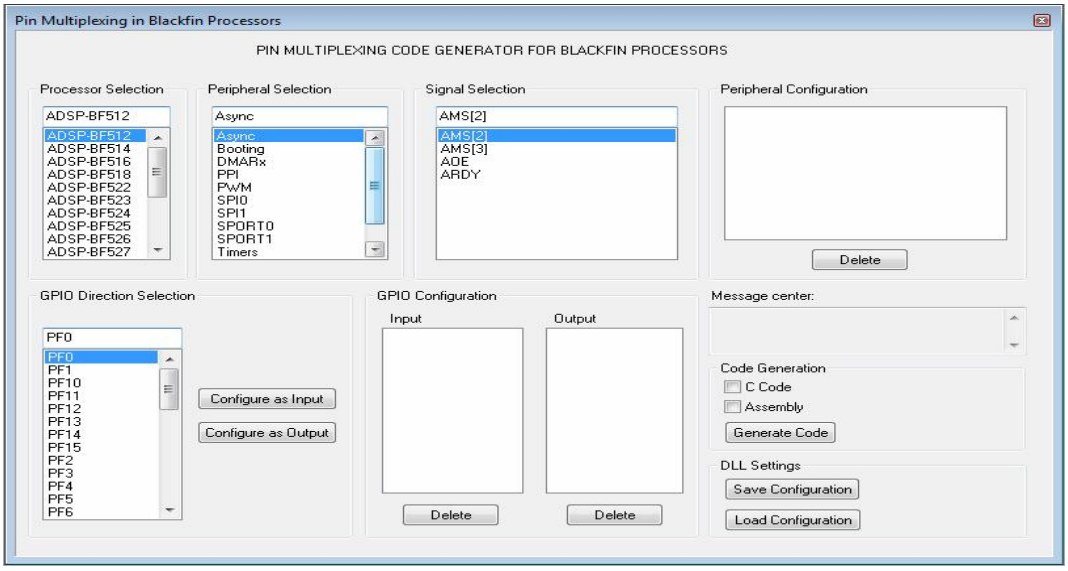

Figure 1. Expert Pin Multiplexing window

Using the Expert Pin Multiplexing Plug-in

To generate code:

1. In `Processor Selection`, select the processor for which code is to be generated.
2. Under `Peripheral Selection`, select the peripheral module of interest.

   When a peripheral module is selected, the signals that appear in the `Signal Selection` list box are updated with all the relevant signals that correspond to the selected peripheral module.
3. To add a peripheral signal, select that particular signal in the `Signal Selection` list box and double-click on it.

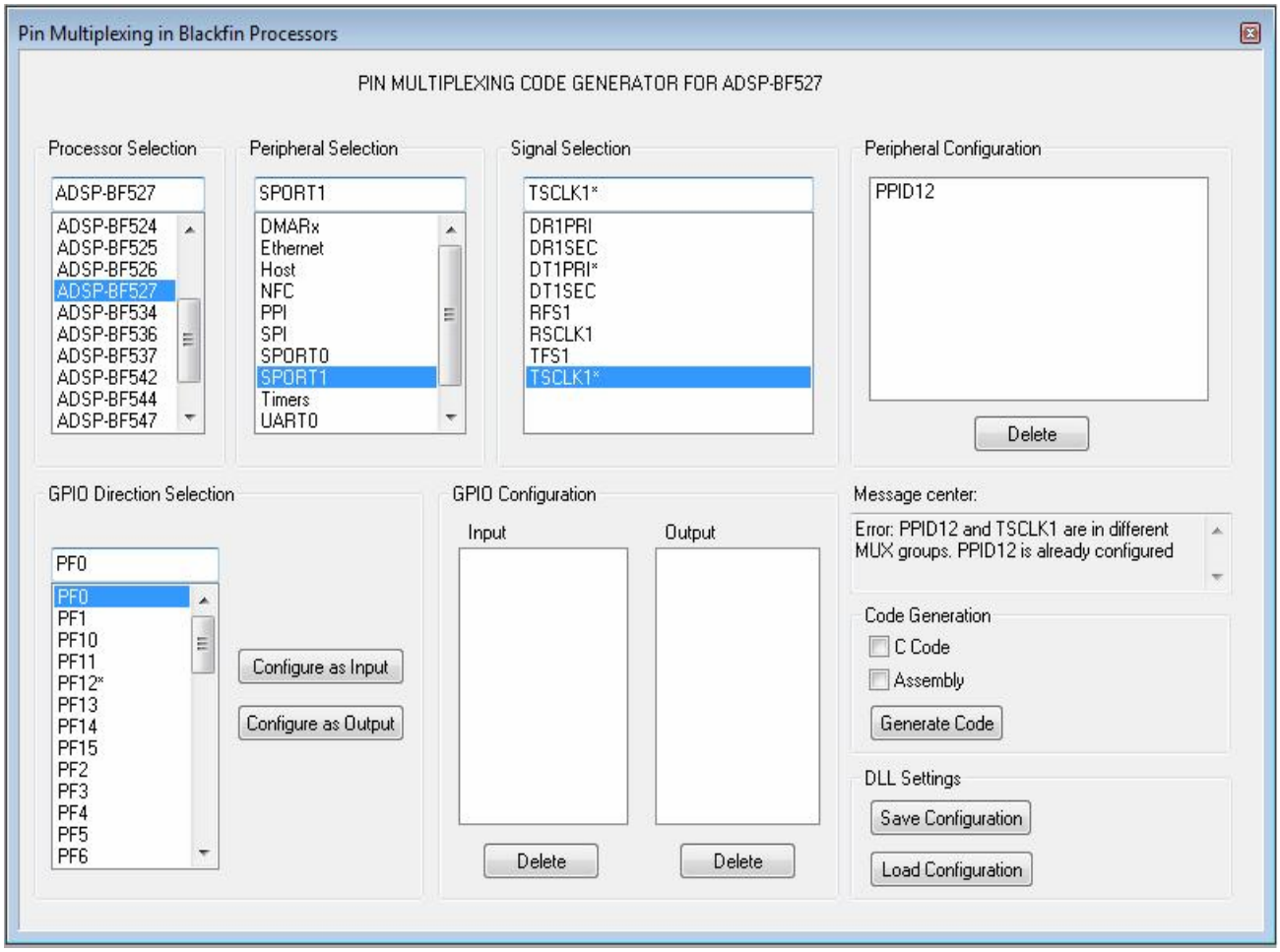

Figure 2. Message center displaying details about the error in configuring a restricted peripheral signal

As peripheral signals are added, the `Peripheral Configuration` list box will be updated appropriately. Whenever a peripheral signal is added, all signals (peripheral/GPIO) that are multiplexed with the configured signal (including the configured signal) will be restricted from being configured later by the tool. The tool provides a visual indication of this restriction by appending all such signal names with '`*`'. For example, consider a scenario where `PPID12` is configured for an ADSP-BF527 processor. Since `PPID12` is multiplexed with `DT1PRI`, `SPISEL2`, and `PF12`, all four signal names will be appended with '`*`'. Note that the `TSCLK1` and `SPISEL3` signal names will also be

appended with '*' even though PPID13 is not configured. This is because both signals (PPID12 and PPID13) belong to the same multiplexing group. The Message center box will display an appropriate error message when a restricted signal is being configured. Figure 2 shows the error message displayed in the Message center box when trying to configure TSCLK1* after PPID12 has already been configured.

4. The GPIO pin can be selected from the GPIO Direction Selection list box. Click the Configure as Input button or Configure as Output button to set the GPIO pin as input or output, respectively.

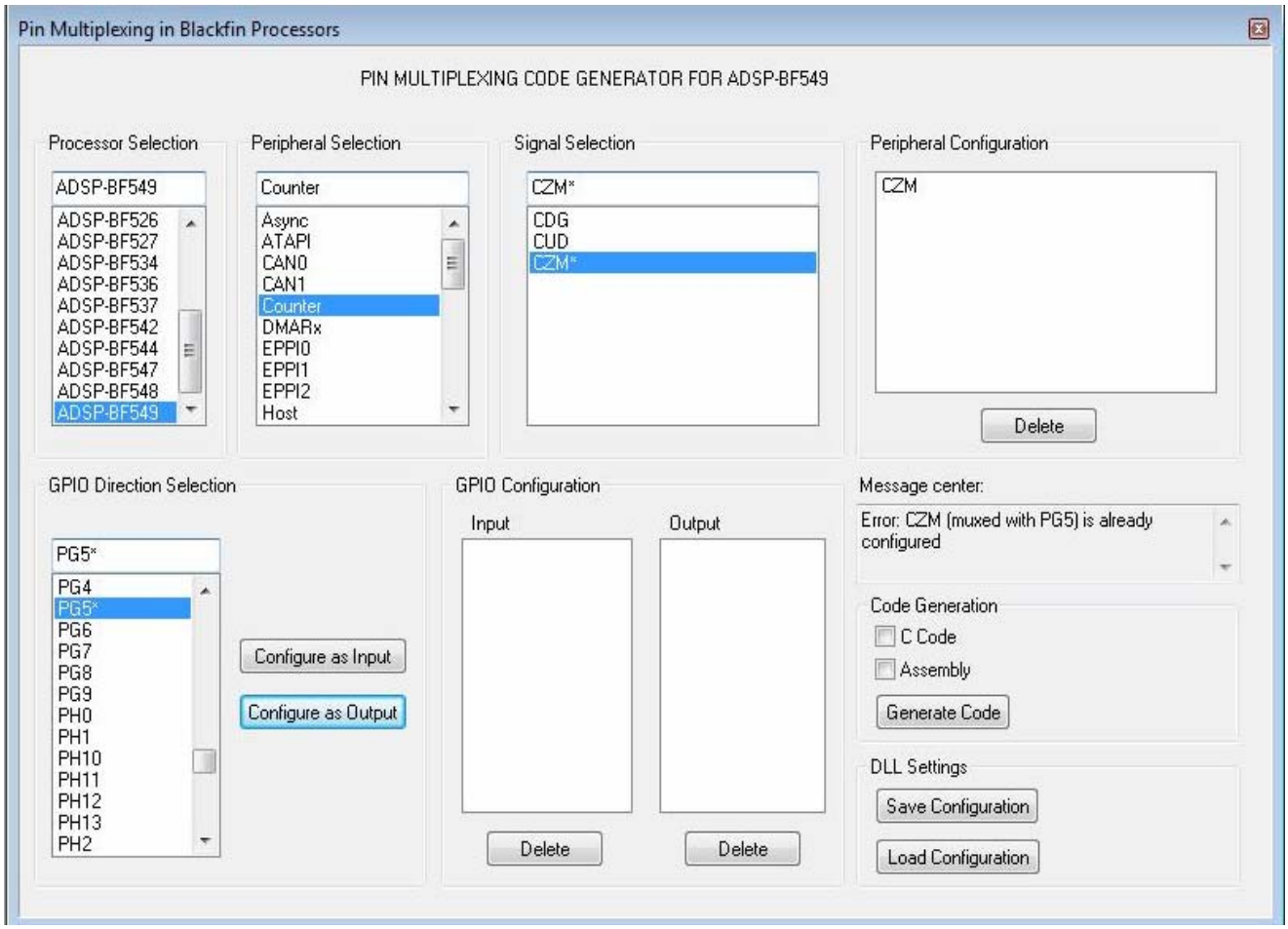

Figure 3. Message center displaying details about the error in configuring a restricted GPIO signal

As GPIO signals are added, the GPIO Configuration list box will be updated appropriately. Whenever a GPIO signal is added, the tool restricts all signals (peripheral/GPIO) that are multiplexed with the configured GPIO pin (including the GPIO pin) from being configured later. The tool also restricts the GPIO pin from being configured as an output if the GPIO pin is already configured as an input, and vice versa. For example, consider a scenario where CZM is configured for an ADSP-BF549 processor. Since CZM is multiplexed with SPI1SEL1, HOST_CE, PPI2_FS2, and PG5, all five signal names will be appended with '*'. Figure 3 shows the error message displayed in the Message center box when PG5* is being configured as an output signal after CZM is configured.

5. Repeat Steps 2 to 4 to add all relevant peripheral signals/GPIOs for your application/system design. Note that there is no restriction in the order in which peripheral/GPIO signals need to be configured.
6. Select the appropriate check boxes and click the `Generate Code` button to generate the C and/or assembly code. This opens a dialog box requesting you to select the path and file names to save the generated C and/or assembly codes. By default, the generated code is named `pin.c` and `pin.asm`.

If the processor type is changed later, data in `Peripheral Configuration` and `GPIO Configuration` is cleared automatically. At the same time, the signals that appear in the `Peripheral Selection`, `Signal Selection`, and `GPIO Direction Selection` boxes are refreshed and updated for the selected processor.

To remove a configured peripheral signal or GPIO input/output pin, click the appropriate `Delete` button after selecting the signal to be deleted. All the list boxes will be updated accordingly.

The Expert Pin Multiplexing plug-in also provides these features:

- Saving a configuration. Clicking the `Save Configuration` button saves the information about the currently selected processor/peripheral/signal/GPIO, the configured peripheral signals/GPIOs, the content of the `Message center` box, and the state of each list box and check box. The information is saved in an output file with a `.cfg` extension (`pin.cfg` is the default name used).
- Loading a configuration. Clicking the `Load Configuration` button loads a saved configuration (`.cfg` file). You are prompted to select a `.cfg` file. After selecting the `.cfg` file, the `Expert Pin Multiplexing` window refreshes, presenting the contents in the `.cfg` file. At this point, peripheral/GPIO signals can be added/deleted per the new design, and code can be regenerated.

Code Generation

This section uses an example to describe the code generation process. Consider an ADSP-BF522 based application where the following peripheral signals and GPIO configuration is desired:

- Peripherals
  - SPORT1 (`DR1PRI`, `DT1PRI`, `RFS1`, `RSCLK1`, `TFS1`, and `TSCLK1`)
  - UART0 (`UART0RX` and `UART0TX`)
  - HOST (`HOST_ACK`, `HOST_ADDR`, `HOST_CE`, `HOST_RD`, `HOST_WR`, and `HOST_Dx` : x = 0 to 15)
  - SPI in slave mode (`MISO`, `MOSI`, `SCK`, and `SPISS`)
- GPIOs:
  - Inputs (`PF3` and `PF5`)
  - Outputs (`PF1` and `PG10`)

Figure 4 demonstrates the generation of C and assembly code to program the Port registers per the above configuration. Assembly and C code generated for this configuration are shown in Listing 1 and Listing 2 of the Appendix, respectively. Generated C/assembly code can be added later to a VisualDSP++ project. For a C project, the `main` function must call the `InitPorts()`function. For an assembly project, the `main` program must call the `_InitPorts` subroutine.

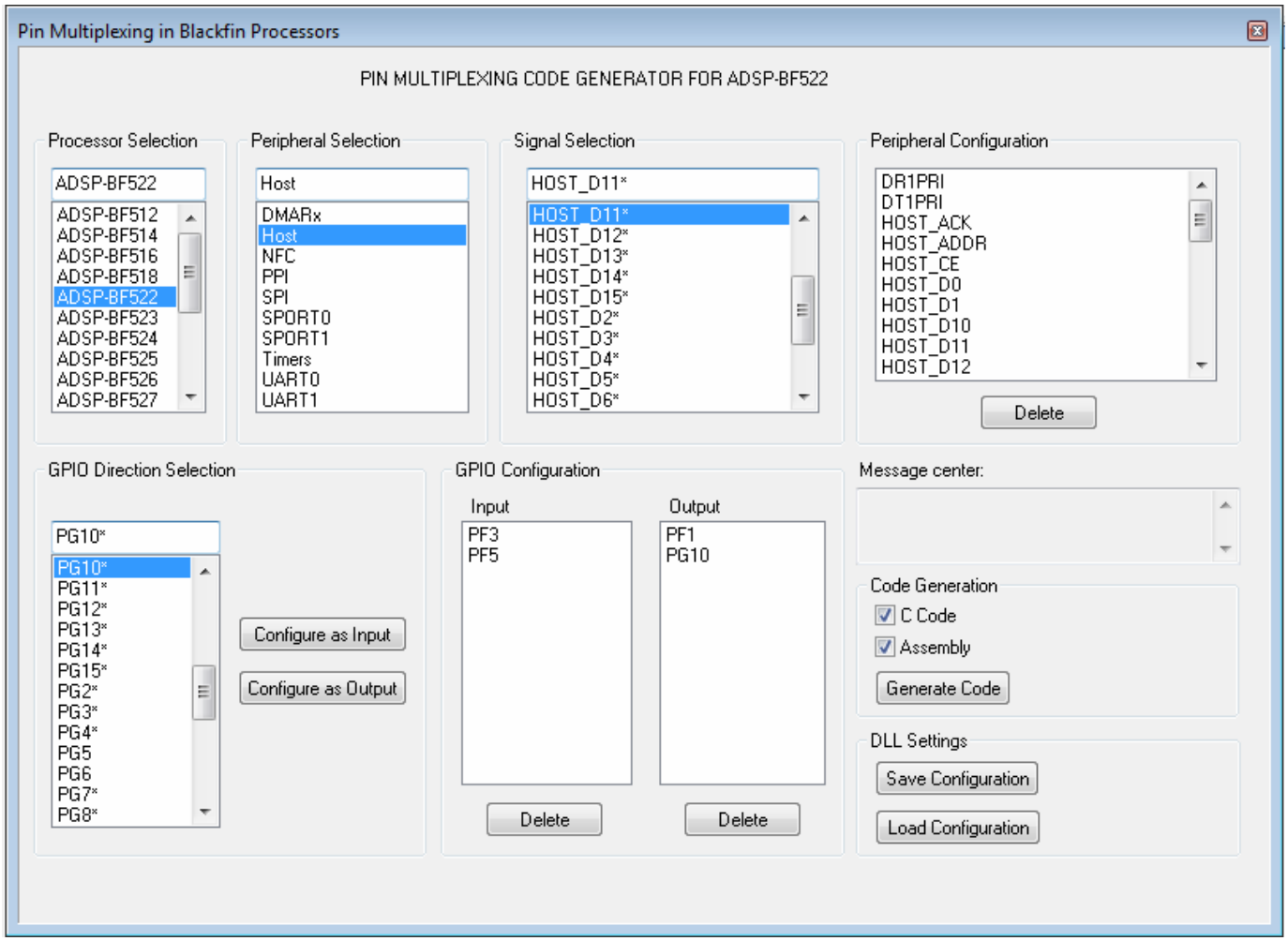

Figure 4. Generating C and assembly codes for configuring the Port registers in the ADSP-BF522 processor

Appendix

pin.asm

```
/*  Assembly code generated to configure PORTs and GPIOs.

    Peripheral pins selected: DR1PRI, DT1PRI, HOST_ACK, HOST_ADDR, HOST_CE, HOST_D0,
HOST_D1, HOST_D10, HOST_D11, HOST_D12, HOST_D13, HOST_D14, HOST_D15, HOST_D2,
HOST_D3, HOST_D4, HOST_D5, HOST_D6, HOST_D7, HOST_D8, HOST_D9, HOST_RD, HOST_WR,
MISO, MOSI, RFS1, RSCLK1, SCK, SPISS, TFS1, TSCLK1, UART0RX, UART0TX

    GPIOs configured as Inputs: PF3, PF5

    GPIOs configured as Outputs: PF1, PG10
*/

#include <defBF52x_base.h>

// This function will setup the Port Control Registers
.section program ;
.global _InitPorts ;

_InitPorts :
    // First save registers
    [--sp] = rets;
    [--sp] = p0;
    [--sp] = r0;

    // PORTx_MUX registers
    p0.l = lo(PORTF_MUX);
    p0.h = hi(PORTF_MUX);
    r0.l = 0x154;
    w[p0] = r0;

    p0.l = lo(PORTG_MUX);
    p0.h = hi(PORTG_MUX);
    r0.l = 0x2820;
    w[p0] = r0;

    p0.l = lo(PORTH_MUX);
    p0.h = hi(PORTH_MUX);
    r0.l = 0x2a;
    w[p0] = r0;

    // PORTx_FER registers
    p0.l = lo(PORTF_FER);
    p0.h = hi(PORTF_FER);
    r0.l = 0x3f00;
    w[p0] = r0;

    p0.l = lo(PORTG_FER);
    p0.h = hi(PORTG_FER);
    r0.l = 0xf99e;
    w[p0] = r0;
```

```
    p0.l = lo(PORTH_FER);
    p0.h = hi(PORTH_FER);
    r0.l = 0xffff;
    w[p0] = r0;

    // PORTxIO_DIR registers
    p0.l = lo(PORTFIO_DIR);
    p0.h = hi(PORTFIO_DIR);
    r0.l = 0x2;
    w[p0] = r0;

    p0.l = lo(PORTGIO_DIR);
    p0.h = hi(PORTGIO_DIR);
    r0.l = 0x400;
    w[p0] = r0;

    p0.l = lo(PORTHIO_DIR);
    p0.h = hi(PORTHIO_DIR);
    r0.l = 0x0;
    w[p0] = r0;

    // PORTxIO_INEN registers
    p0.l = lo(PORTFIO_INEN);
    p0.h = hi(PORTFIO_INEN);
    r0.l = 0x28;
    w[p0] = r0;

    p0.l = lo(PORTGIO_INEN);
    p0.h = hi(PORTGIO_INEN);
    r0.l = 0x0;
    w[p0] = r0;

    p0.l = lo(PORTHIO_INEN);
    p0.h = hi(PORTHIO_INEN);
    r0.l = 0x0;
    w[p0] = r0;

    // Restore the registers
    r0 = [sp++];
    p0 = [sp++];
    rets = [sp++];

    // Return back from the subroutine
    rts;
```

Listing 1. pin.asm

pin.c

```
/*
    C code generated to configure PORTs and GPIOs.

    Peripheral pins selected: DR1PRI, DT1PRI, HOST_ACK, HOST_ADDR, HOST_CE, HOST_D0,
HOST_D1, HOST_D10, HOST_D11, HOST_D12, HOST_D13, HOST_D14, HOST_D15, HOST_D2,
HOST_D3, HOST_D4, HOST_D5, HOST_D6, HOST_D7, HOST_D8, HOST_D9, HOST_RD, HOST_WR,
MISO, MOSI, RFS1, RSCLK1, SCK, SPISS, TFS1, TSCLK1, UART0RX, UART0TX

    GPIOs configured as Inputs: PF3, PF5

    GPIOs configured as Outputs: PF1, PG10
*/

#include <cdefBF52x_base.h>

void InitPorts();

// This function will setup the Port Control Registers
void InitPorts()
{
    // First Set PORTx_MUX registers
    *pPORTF_MUX = 0x154;
    *pPORTG_MUX = 0x2820;
    *pPORTH_MUX = 0x2a;

    // Set PORTx_FER registers
    *pPORTF_FER = 0x3f00;
    *pPORTG_FER = 0xf99e;
    *pPORTH_FER = 0xffff;

    // Set PORTxIO_DIR registers
    *pPORTFIO_DIR = 0x2;
    *pPORTGIO_DIR = 0x400;
    *pPORTHIO_DIR = 0x0;

    // Set PORTxIO_INEN registers
    *pPORTFIO_INEN = 0x28;
    *pPORTGIO_INEN = 0x0;
    *pPORTHIO_INEN = 0x0;
}
```

Listing 2. pin.c

References

[1] *ADSP-BF51x Blackfin Processor Hardware Reference Preliminary*, Rev 0.1, January 2009. Analog Devices, Inc.

[2] *ADSP-BF52x Blackfin Processor Hardware Reference (Volume 1 of 2).* Rev 0.31 (Preliminary), May 2008. Analog Devices, Inc.

[3] *ADSP-BF537 Blackfin Processor Hardware Reference*, Rev 3.2, March 2009. Analog Devices, Inc.

[4] *ADSP-BF54x Blackfin Processor Hardware Reference (Volume 1 of 2) Preliminary.* Rev 0.4, August 2008. Analog Devices, Inc.

[5] *ADSP-BF512/BF514/BF516/BF518(F) Blackfin Embedded Processor Preliminary Data Sheet.* Rev PrH, November 2009. Analog Devices, Inc.

[6] *ADSP-BF522/BF523/BF524/BF525/BF526/BF527 Blackfin Embedded Processor Preliminary Data Sheet.* Rev A, November 2009. Analog Devices, Inc.

[7] *ADSP-BF534/BF536/BF537 Blackfin Embedded Processor Preliminary Data Sheet.* Rev G, February 2009. Analog Devices, Inc.

[8] *ADSP-BF542/BF544/BF547/BF548/BF549 Blackfin Embedded Processor Preliminary Data Sheet.* Rev B, February 2009. Analog Devices, Inc.

Document History

Revision	Description
Rev 2 – January 7, 2010 *by R. Jagadeesh and K. Anand*	Updated the document and the plug-in to include support for ADSP-BF51x and ADSP-BF534/6/7 Blackfin Processors.
Rev 1 – May 15, 2008 *by R. Jagadeesh*	Initial release.