



## Expert Pin Multiplexing Plug-in for Blackfin® Processors

Contributed by Jagadeesh R, Anand K and Prashant G

Rev 3 – July 22, 2011

### 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-BF50x, ADSP-BF51x, ADSP-BF52x, ADSP-BF534/BF536/BF537 (hereafter referred to as ADSP-BF537 processors), ADSP-BF54x and ADSP-BF592 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. Most of the associated pins are shared by multiple signals. The ports function as multiplexer controls.

The ADSP-BF50x and 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, these are grouped into ten ports – referred to as Port A through Port J. While in ADSP-BF592 processor, these are grouped into two ports – Port-F and Port-G.


Almost all pins (except Port J pins of BF52x and BF537 processors) can also function as a General Purpose Input/Output (GPIO) pin.

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  $x = F, G, \text{ or } H$  for ADSP-BF50x, ADSP-BF51x, ADSP-BF52x, and ADSP-BF537 processors;  $x = A \text{ to } J$  for ADSP-BF54x processor; or  $x = F, G$  for ADSP-BF592 processor).


The competing peripherals on Ports are controlled by the respective multiplexer control register (`PORTx_MUX` for ADSP-BF50x, ADSP-BF51x, ADSP-BF52x, ADSP-BF54x, and ADSP-BF592 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 a pin a digital GP input pin, the input driver must be enabled in the `PORTxIO_INEN` register (`PORTx_INEN` for ADSP-BF54x processors).


 The input signals in the “Additional Use” column are enabled by their module only, regardless of the state of the PORTx\_MUX and PORTx\_FER registers. E.g. Rotary Counter pins in ADSP-BF50x processors are featured under the ‘Additional Use’ column.

By default, all pins are in GPIO mode and are configured as inputs after reset. Neither GPIO output nor input drivers are active by default. However, GPIO input drivers are disabled to minimize power consumption and any need for external pull-up/pull-down resistors on unused.

 For more details on pin multiplexing please refer ‘General Purpose PORTs’ chapter of the *Hardware Reference Manual* of the respective processor <sup>[1][2][3][4][5][6]</sup>.

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 requiring much information about internal details.

 The signal names used in the plug-in may differ from the actual pin names of the processor. Please refer to the *Datasheet* of the respective processor <sup>[7][8][9][10][11][12]</sup>.

## 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 and Windows 7 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-BF50x, ADSP-BF51x, ADSP-BF52x, ADSP-BF537, ADSP-BF54x, and ADSP-BF592 sessions in VisualDSP++ release 5.0 or higher.

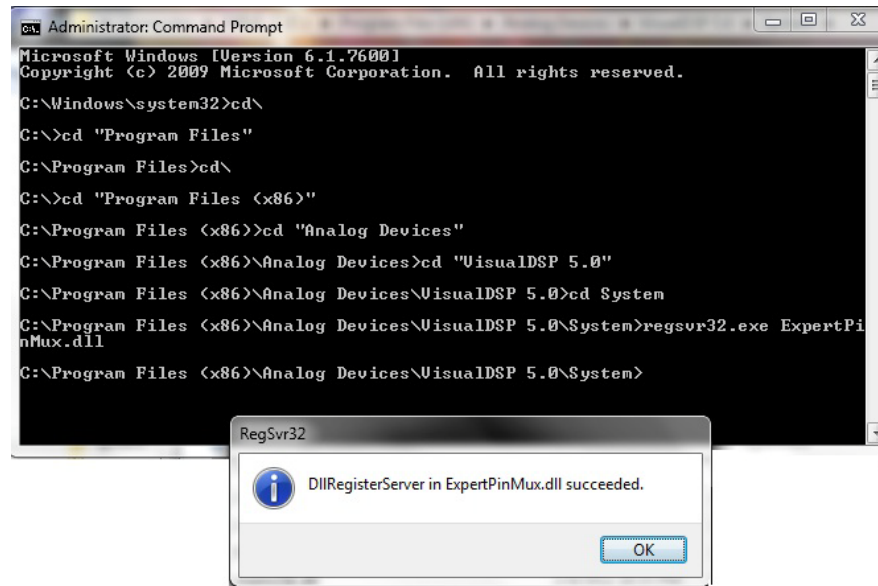


Figure 1. Installing the Expert Pin Multiplexing plug-in

Figure 2 shows the default state of the Expert Pin Multiplexing window. By default, the ADSP-BF504 processor is selected and all the list boxes are populated accordingly.

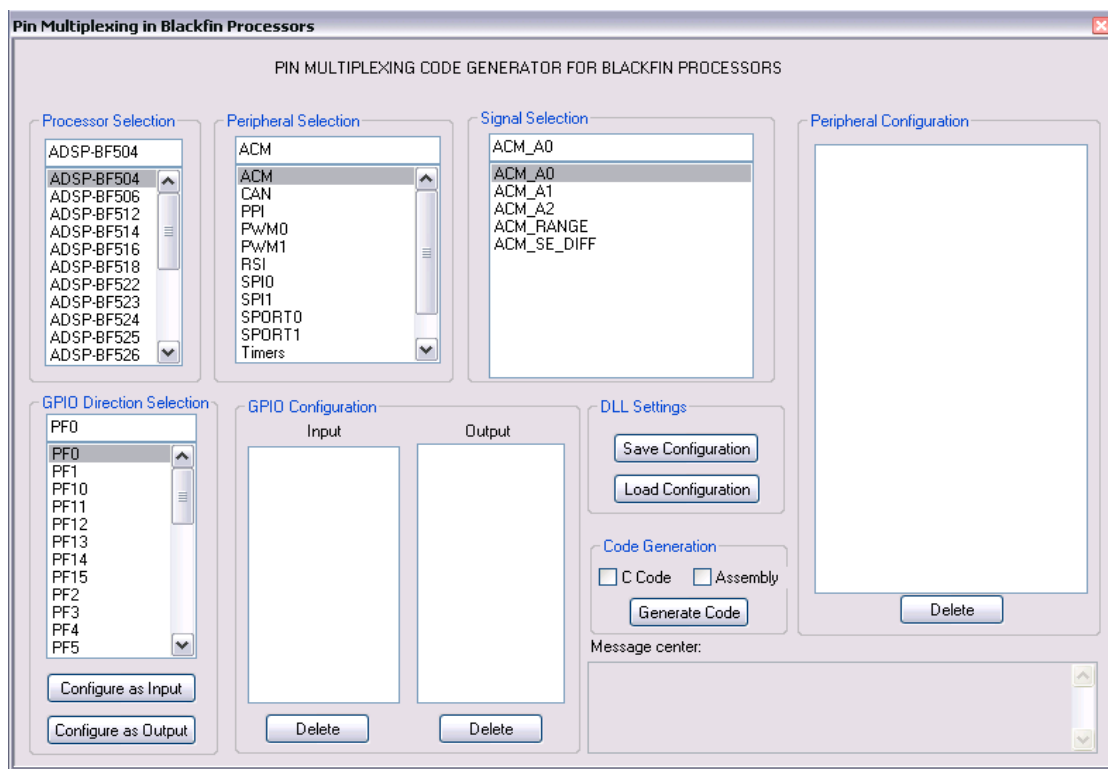


Figure 2. Expert Pin Multiplexing window

## 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.

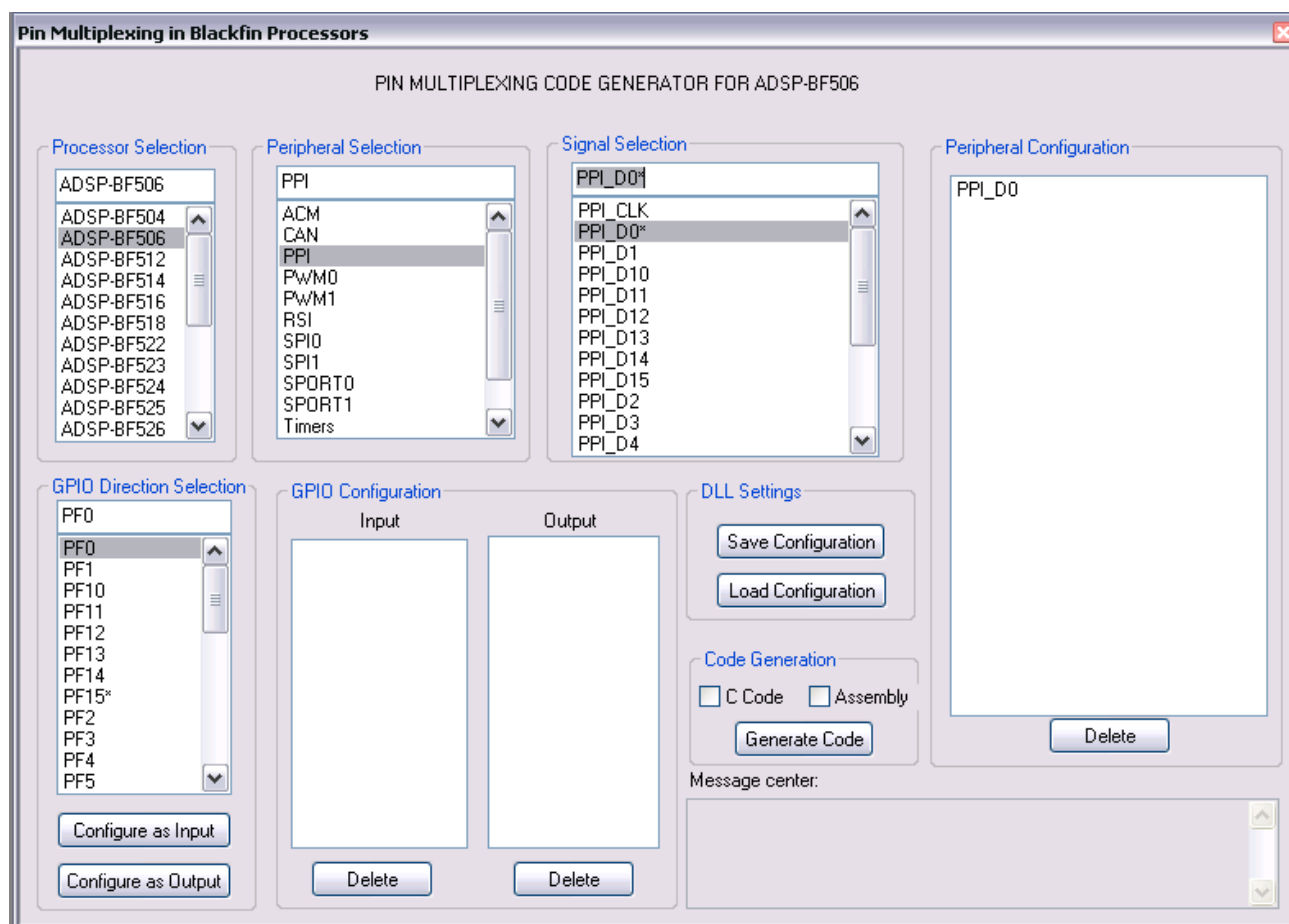


Figure 3. Adding peripheral signals

4. 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 the above example where PPI\_D0 is configured for an ADSP-BF506 processor. Since PPI\_D0 is multiplexed with PWM0\_AL and SPI0\_SSEL3:PF15 and PF15, these three signal names along with PPI\_D0 signal name will be appended with '\*'.

Note that the PWM0\_AH and SPI0\_SSEL2 signal names will also be appended with '\*' even though PPI\_D1 or PF14 is not configured. This is because PPI\_D0 and PPI\_D1 signals belong to the same multiplexing group. (Obviously, in this case PF14 can be configured as GPIO).

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 PWM0\_AH\* after PPI\_D0 has already been configured.

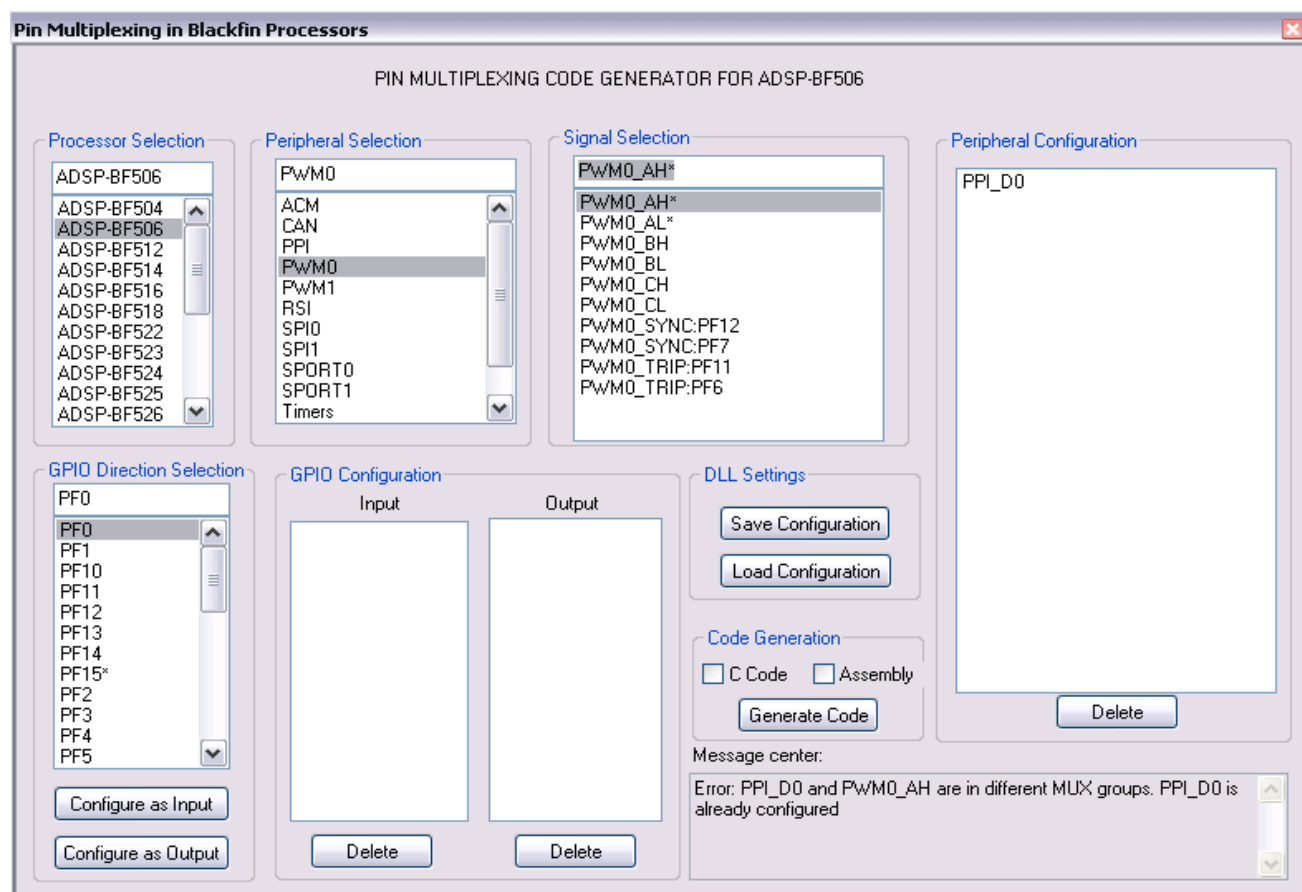


Figure 4. Message center displaying details about the error in configuring a restricted peripheral signal

5. 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.

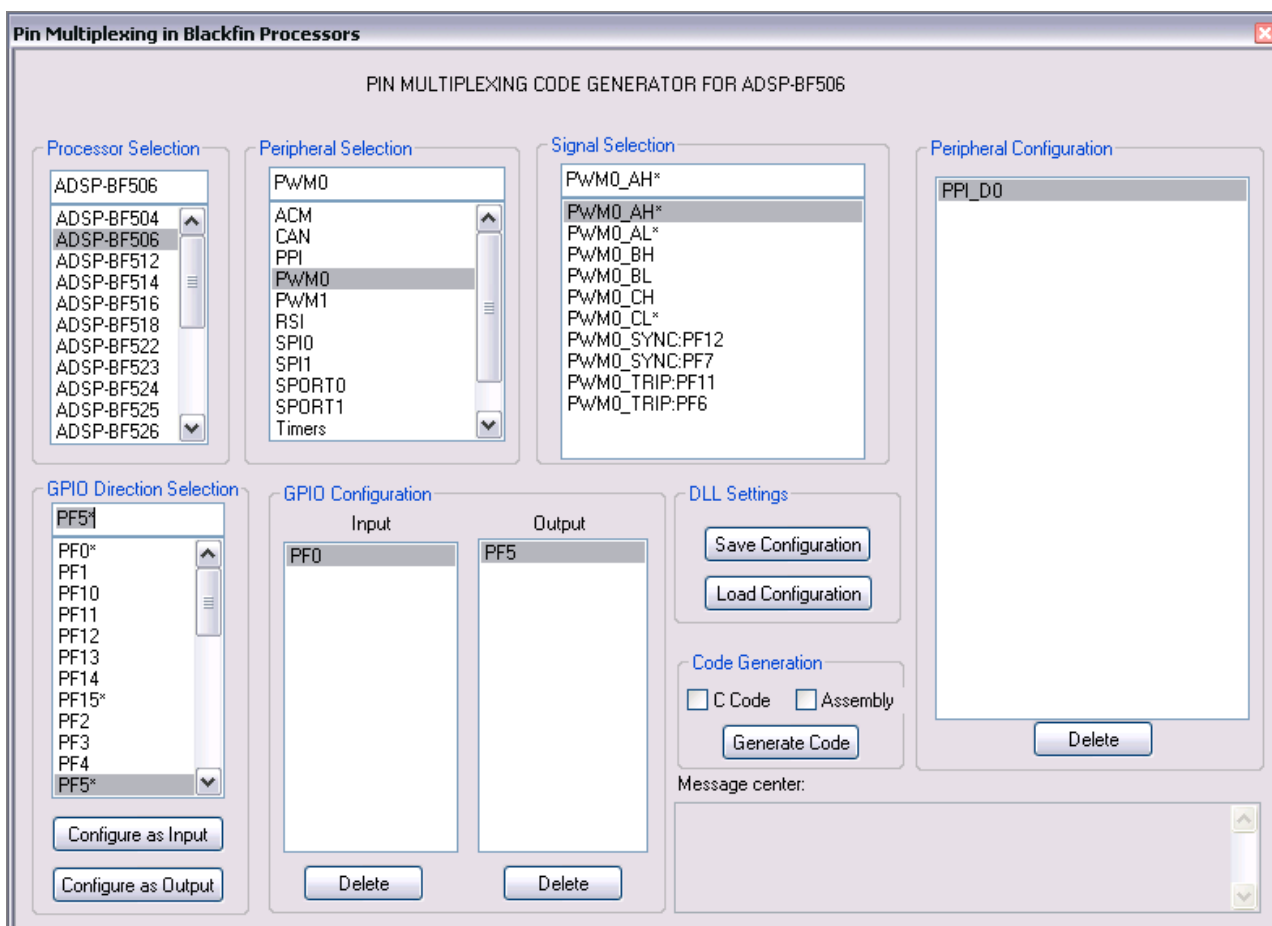


Figure 5. Configuring pins as GPIOs

As GPIO signals are added, the GPIO Configuration list box will be updated appropriately. As in case of peripheral signals, 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, in the above example, PF0 is configured as GP input while PF5 is configured as GP output. Since PF0 is multiplexed with TSCLK0, UART0\_RX:PF0 and TMR6 peripheral signals, all these signal names will be appended with '\*'. Similarly only signal multiplexed with PF5 i.e. DR0PRI, PWM0\_CL and PPI\_D11 will be appended with '\*'. After this these signals will be restricted.

6. Repeat steps 2 to 5 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.
7. 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.
8. Select the appropriate check boxes to generate the C and/or assembly code and click the Generate Code button. 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.

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 (`TSCLK1`, `TFS1`, `DT1PRI`, `RSCLK1`, `RFS1` and `DR1PRI`)
  - UART0 (`UART0_RX` and `UART0_TX`)
  - HOST (`HOST_ACK`, `HOST_ADDR`, `HOST_CE`, `HOST_RD`, `HOST_WR`, and `HOST_Dx` : `x = 0 to 15`)
  - SPI in slave mode (`SPI_MISO`, `SPI_MOSI`, `SPI_SCK`, and `SPI_SS`)
- GPIOs:
  - Inputs (`PF3` and `PF5`)
  - Outputs (`PF1` and `PG10`)

Figure 6 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.

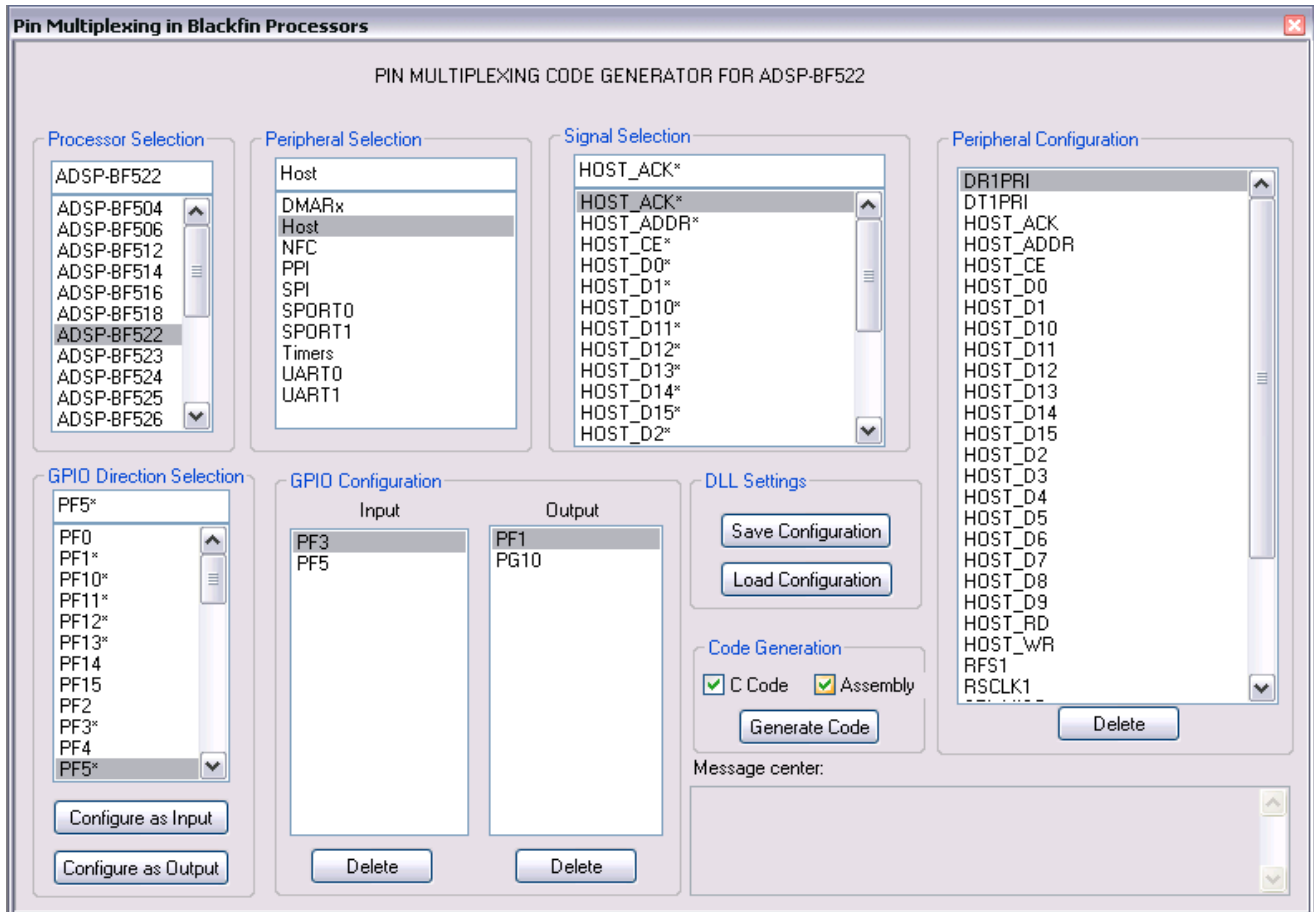


Figure 6. Generating C and assembly codes



## 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,
RFS1, RSCLK1, SPI_MISO, SPI_MOSI, SPI_SCK, SPI_SS, TFS1, TSCLK1, UART0_RX, UART0_TX

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;

    // point P0 to system MMR space
    P0.H = 0xFC00;
    P0.L = 0x0000;
    R0 = 0x0000;

    // PORTx_FER registers
    R0.L = 0x3f00;
    w[P0 + LO(PORTF_FER)] = R0;

    R0.L = 0xf99e;
    w[P0 + LO(PORTG_FER)] = R0;

    R0.L = 0xffff;
    w[P0 + LO(PORTH_FER)] = R0;

    //-----

    // PORTx_MUX registers
    R0.L = 0x154;
    w[P0 + LO(PORTF_MUX)] = R0;

    R0.L = 0x2820;
    w[P0 + LO(PORTG_MUX)] = R0;

    R0.L = 0x2a;
    w[P0 + LO(PORTH_MUX)] = R0;
```

```
//-----  
  
// PORTxIO_DIR registers  
R0.L = 0x2;  
w[P0 + LO(PORTFIO_DIR)] = R0;  
  
R0.L = 0x400;  
w[P0 + LO(PORTGIO_DIR)] = R0;  
  
R0.L = 0x0;  
w[P0 + LO(PORThIO_DIR)] = R0;  
  
//-----  
  
// PORTxIO_INEN registers  
R0.L = 0x28;  
w[P0 + LO(PORTFIO_INEN)] = R0;  
  
R0.L = 0x0;  
w[P0 + LO(PORTGIO_INEN)] = R0;  
  
R0.L = 0x0;  
w[P0 + LO(PORThIO_INEN)] = R0;  
  
//-----  
  
// Restore the registers  
R0 = [SP++];  
P0 = [SP++];  
RETS = [SP++];  
  
// Return back from the subroutine  
_InitPorts.END: 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,
 RFS1, RSCLK1, SPI_MISO, SPI_MOSI, SPI_SCK, SPI_SS, TFS1, TSCLK1, UART0_RX, UART0_TX

 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-BF50x Blackfin Processor Hardware Reference*, Rev 1.0, December 2010, Analog Devices, Inc.
- [2] *ADSP-BF51x Blackfin Processor Hardware Reference*, Revision 1.0, September 2010, Analog Devices, Inc.
- [3] *ADSP-BF52x Blackfin Processor Hardware Reference*, Rev 1.0, March, 2010, Analog Devices, Inc.
- [4] *ADSP-BF537 Blackfin Processor Hardware Reference*, Rev 3.2, March 2009, Analog Devices, Inc.
- [5] *ADSP-BF54x Blackfin Processor Hardware Reference*, Rev 1.0, August 2010, Analog Devices, Inc.
- [6] *ADSP-BF59x Blackfin Processor Hardware Reference*, Rev 1.0, May 2011, Analog Devices, Inc.
- [7] *ADSP-BF504/ADSP-BF504F/ADSPBF506F: Blackfin Embedded Processor Data Sheet*, Rev 0, December 2010, Analog Devices, Inc.
- [8] *ADSP-BF512/ADSP-BF514/ADSP-BF516/ADSP-BF518(F) Blackfin Embedded Processor Data Sheet*, Rev. B, February 2011, Analog Devices, Inc.
- [9] *ADSP-BF522/ADSP-BF523/ADSP-BF524/ADSP-BF525/ADSP-BF526/ADSP-BF527: Blackfin Embedded Processor Data Sheet*, Rev B, May 2010, Analog Devices, Inc.
- [10] *ADSP-BF534/ADSP-BF536/ADSP-BF537: Blackfin Embedded Processor Data Sheet*, Rev I, July 2010, Analog Devices, Inc.
- [11] *ADSP-BF542/ADSP-BF544/ADSP-BF547/ADSP-BF548/ADSP-BF549 Blackfin Embedded Processor Data Sheet*, Rev D, June 2011, Analog Devices, Inc.
- [12] *ADSP-BF592 Blackfin Embedded Processor Preliminary Data Sheet*, Rev 0, May 2011, Analog Devices, Inc.

## Document History

Revision	Description
<i>Rev 3 – July 22, 2011 by Prashant G.</i>	Updated the document and the plug-in to include support for ADSP-BF50x and ADSP-BF592 Blackfin Processors.
<i>Rev 2 – January 7, 2010 by R. Jagadeesh and K. Anand</i>	Updated the document and the plug-in to include support for ADSP-BF51x and ADSP-BF534/6/7 Blackfin Processors.
<i>Rev 1 – May 15, 2008 by R. Jagadeesh</i>	Initial release.