

Interfacing the MSP430 With a DSP Application

Randy Wu

AEC MSP430 Products

ABSTRACT

The MSP430 is typically used in applications such as metering, intelligent sensing, control systems and portable instrumentation. In some systems, the microcontroller may need to interface to an external device such as a DSP co-processor to handle digital signal processing-intensive calculations and operations. This application note describes and provides a complete hardware and software reference design which includes frameworks running on both the MSP430 and DSP, as well as incorporating Inter Processor Communication (IPC) drivers to implement a simple message-passing scheme between the two processors. A Digital Audio Player is shown as an example. Supplied with this application note are complete reference frameworks for the MSP430 and DSP which can both be run out of the box and then subsequently adapted for other real-world system designs. The complete projects and source code are provided as separate packages for both the MSP430 and TMS320 platforms.

Contents

1	Introduction	2
2	Exercising the MSP430 Hardware User Interface.....	3
3	The MSP430-Side Reference Framework	3
4	The DSP-Side Reference Framework	5
5	Conclusion.....	6
6	References	6
Appendix A	Connecting UART, DSP, and MSP430 Development Boards	7
Appendix B	TMS320VC5510 Software UART Driver via McBSP	9
Appendix C	Connecting the MSP430 to the C5510 Enhanced HPI (EHPI)	10
Appendix D	Creating a Stand-Alone Embedded System Using FlashBurn™	13

List of Figures

1	Hardware Connections and System Block Diagram	3
2	Message Passing using MSP430 and DSP UART Link.....	5
C-1	MSP LINK Using DSP Enhanced Host Port Interface (EHPI)	10

List of Tables

1	Responsibilities of the MSP430 and DSP	2
C-1	Pin Connections Between ES449 and DSK5510	11
C-2	Pull-Down Resistors for DSK5510 EHPI.....	11
C-3	Internal Connections for DSK5510 HPI in Multiplexed Mode	11
D-1	File and Directory Names for Restoring the C5510DSK's POST Flash Program	14

1 Introduction

In this reference design, an Audio Player has been developed as a sample application to demonstrate user and system control maintained by the MSP430 while the DSP applies a volume control algorithm to an audio input stream and produces a stereo audio output to the outside world.

In terms of real-world applications, the tasks handled by the MSP430 and TMS320 can be divided as shown in [Table 1](#).

Table 1. Responsibilities of the MSP430 and DSP

MSP430 MICROCONTROLLER	TMS320 DSP
Possible User Interface Functions: <ul style="list-style-type: none"> • Button decoding • LCD Display • Temperature sensing • Low battery detection 	Potential Signal Processing Functions: <ul style="list-style-type: none"> • Audio Encoding, Decoding, Enhancement, Filtering, Special Effects • Stereo playback via D/A Converters • Speech Recognition and Synthesis

In order for both devices to work in harmony, there must be some form of Inter Processor Communication (IPC) between the two devices during real-time operation of the system. For this reference design, a UART serial link between the two processors is used for sending data back and forth. The DSP can process each message in real-time and carry out the commands of each message sent by the MSP430. In this application note, this UART-based device driver will be referred to as MSP Link. Other alternatives for the physical connection between the MSP430 and DSP are discussed in the Appendices.

The source code is written in C language to provide portability, readability, maintainability, and reusability. The device driver itself is packaged in a modular style so that it can be easily reused in applications requiring a UART-based serial link between the MSP430 and a TMS320 DSP.

Two reference frameworks are provided as an example to demonstrate how the device driver on each end is used in an MSP430+DSP system. A TMS320C5510™ DSP Starter Kit from Spectrum Digital for the DSP hardware and an ES449 demo board from SoftBaugh for the MSP430 hardware are used for this example application. This allows the system developer to get started quickly.

The following hardware components must be obtained to run the featured project:

- Spectrum Digital TMS320VC5510™ DSP Starter Kit (DSK)
- DSPGlobal IS-232 UART RS232/422 Daughtercard (plugs directly onto TI DSK's) with a standard (straight-through) DB-9 Serial Cable
- SoftBaugh ES449 Demo Board
- MSP430 Flash Emulation Tool (FET) or equivalent JTAG Emulator
- A stereo output device with 3,5 mm input connector example: Dual PC speakers or headphones
- Audio input source for DSK board example: Host PC playing MP3/WAV/MIDI file with soundcard output, CD player, portable MP3 player, etc.

The detailed procedure on how to connect the various hardware components can be found in [Appendix A](#). In addition to describing the hardware UART configuration which involves connecting the 3 boards (UART, MSP430, DSP) together, two other interprocessor connections are discussed in the subsequent Appendices. In summary, this application note discusses 3 configurations:

1. MSP430 USART ↔ Hardware UART ↔ TMS320 DSP External Memory Interface (EMIF) [[Appendix A](#)]
2. MSP430 USART ↔ TMS320 DSP MultiChannel Buffered Serial Port (McBSP) emulating a Software UART [[Appendix B](#)]
3. MSP430 GPIO ↔ TMS320 DSP Enhanced Host Port Interface (EHPI) [[Appendix C](#)]

Note:

Complete projects are provided with this application note for Configurations 1 and 2. Sample source code is provided for Configuration #3, in addition, the sample project provided for the MSP430 USART will need to be modified accordingly. Refer to the respective Appendix for additional detailed descriptions for each configuration.

2 Exercising the MSP430 Hardware User Interface

Once both the DSP and MSP430 applications are up and running on their respective development boards, the user can use both pushbutton switches (SW1 and SW2) on the ES449 demo board to control the Audio Player and get feedback on the operating parameters of the system, such as the current temperature, the current volume setting, and the audio filter ON/OFF/MUTE status of the audio output stream. The LCD display is refreshed periodically with the current temperature reading as well as the currently selected volume level. At reset, both boards will be initialized for a volume setting of 0, so be sure to increase the volume of the system by repeatedly pressing the SW2 button on the ES449 demo board, and be sure to select filter ON/OFF mode (volume must be greater than '00') by pressing the SW1 and SW2 buttons on the ES449 demo board, respectively. Using SW1 to select **ON** will enable the FIR filter while **OFF** disables FIR filtering; when **MUTE** is selected, the volume will be set to 0 on the DSP.

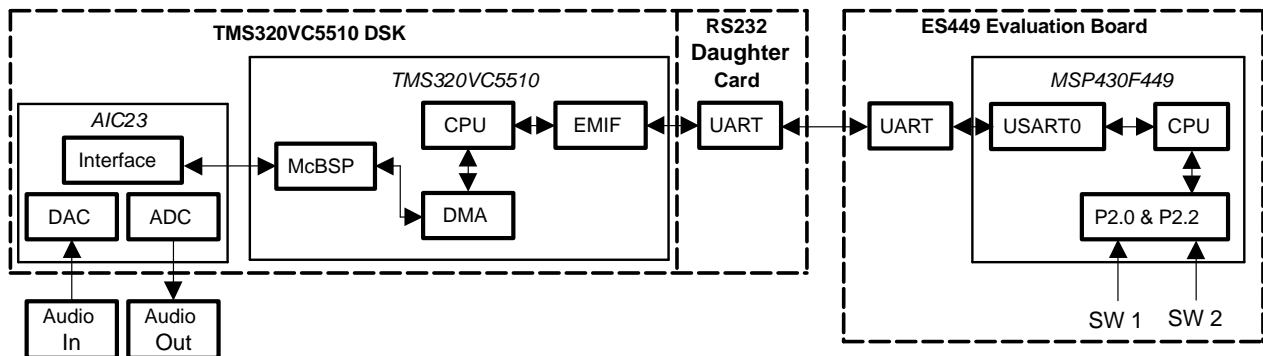


Figure 1. Hardware Connections and System Block Diagram

3 The MSP430-Side Reference Framework

3.1 MSP430 Real-Time Operating System: Salvo™ O.S.

The MSP430-side framework is based upon and built on top of the Salvo™ Real-Time O.S. provided by a TI Third Party, Pumpkin Inc. Salvo™ is a Real-Time Operating System (RTOS) designed expressly for very-low-cost embedded systems. Typical applications use 1-2K ROM and 50–100 bytes of RAM. More detailed information can be found at www.pumpkininc.com.

Note:

In order to rebuild the provided MSP430 sample project, first download the latest release of the *Salvo™ Demo Version* directly from the Pumpkin Inc. website and install the MSP430-specific build for the SalvoLite™ application. The main installation file should be the latest executable file in the download area. Be sure to install the SalvoLite™ files to the directory <C:\salvo> since the sample project will be expecting the required Salvo™ files to be there. Otherwise some of the Salvo-included files of the project must be manually removed and added back into the project since the IAR Systems EW430 project needs to know where the Salvo™ installation files reside.

3.2 ES449 Demo Board LCD Device Drivers

A complete device driver layer for the ES449 demo board and SBLCDA2T configuration has been provided as part of the MSP430-side reference framework. This highly modular and portable software component was written completely in C to allow it to be reused in actual production software designs. The *es449_sblcda2t* device driver provides the following APIs:

```
void LCD_init(void);
// Usage:      Needs to be called once during system initialization
// Parameters: none

void LCD_putChar(unsigned int val, unsigned char loc);
// Usage:      Display a character in any of the 7 main segments
// Parameters: val = character as defined in "es449_sblcda2t.h"
//            loc = segment number starting from left (1 to 7)

void LCD_putHex(unsigned char val, unsigned char loc);
// Usage:      Display a character in any of the 4 upper right-hand segments
// Parameters: val = character as defined in "es449_sblcda2t.h"
//            loc = segment number starting from left (1 to 4)

void LCD_putSTOP(void);
// Usage:      Display the word STOP in main segments 1-4
// Parameters: none

void LCD_putPLAY(void);
// Usage:      Display the word PLAY in main segments 1-4
// Parameters: none

void LCD_splashScreen(void);
// Usage:      Display the opening banner for the application
// Parameters: none

void LCD_displayTemperature(int temp);
// Usage:      Displays 2-digit temperature in degrees F
// Parameters: temp = any number from 0 to 99

void LCD_displayAccessories(void);
// Usage:      Rotate accessories as an example of animating the screen
// Parameters: none

char LCD_convertNumToCharFor12(char num);
// Usage:      converts decimal number to the corresponding character in "es449_sblcda2t.h"
// Parameters: num = any number from 0 to 9
```

3.3 MSP430 to TMS320 Inter Processor Communication MSP Link

There are a few options to physically connect the MSP430 with a DSP. One way would be for the DSP to use its own MultiChannel Buffered Serial Port (McBSP) to emulate a software UART. Another way would be to use the DSP's Enhanced Host Port Interface (EHPI) to allow the MSP430 to read from and write to a section of the DSP's internal memory as shared mailboxes. The third way would be to connect a hardware UART to the DSP's External Memory Interface (EMIF) to establish a standard serial link to the MSP430's Universal Synchronous Asynchronous Transmitter Receiver (USART). The out-of-the-box project provided with this application note implements the hardware UART method; the appendices will offer some additional guidance to implementing the other 2 methods (software UART and EHPI).

The software component which handles the Inter Processor Communication is implemented as a device driver that treats the DSP as an external device connected to the MSP430. See [Figure 2](#) for the software components which make up a basic message passing scheme by having the MSP430 write commands to the DSP's internal memory via the UART-based serial link. A small section of memory is allocated within the DSP to act as a FIFO queue of message holders which contain the data written by the MSP430. Should the DSP need to send messages back to the MSP430, the UART Interrupt Service Routine is set up to send data back through the serial link as well. Each time a message is sent by either the MSP430 or DSP, an interrupt is generated to alert the other device that a message has just been written to the serial link. Each framework is then able to read the message in an Interrupt Service Routine where it can be copied onto a queue and then subsequently processed by a task function within the RTOS.

For simplicity, the MSP430 Link device driver, out of the box, assumes that each message is 16 bits long (two 8-bit characters). These 16 bits can then be packed into a message when received on the DSP side using the following data structure defined in the "msplink_dsp.h" file:

```

typedef struct MSPLINK_Msg {      // standard message format
    unsigned char cmd; // command code to be executed
    unsigned char arg; // optional argument for the command
} MSPLINK_Msg;
  
```

Of course, it is up to the application designer to determine valid command codes and their respective meanings and arguments. In the Audio Player example, simple commands consist of stop, play, and set volume operations by treating the **arg** field as the current volume to be set. A stop command simply sets the volume to 0, while the actual volume settings are sent to the DSP under the condition that the audio mode is currently set for the play mode.

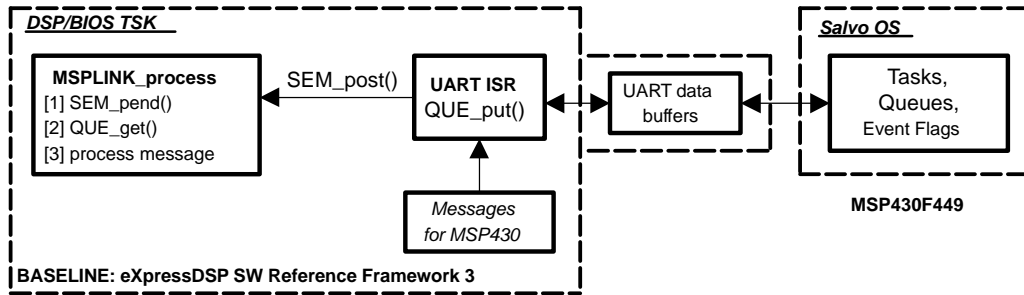


Figure 2. Message Passing using MSP430 and DSP UART Link

3.4 MSP430 Internal Temperature Sensor

The MSP430F449 device contains a built-in temperature sensor integrated into the internal A/D converter. This allows the Audio Player application to take periodic temperature readings and display the current temperature value on the LCD display in the upper right-hand corner. Each time a temperature conversion has completed, an interrupt is generated by the ADC12 module to the CPU. The ADC ISR reads the result register and translates the value into a two-digit temperature reading in degrees Fahrenheit, and then displays the value in the upper right-hand corner of the LCD display. The temperature reading is updated constantly on the LCD.

3.5 User Interface Application Layer

The application consists of a task which periodically toggles an LED and another task which displays the current temperature reading, starts another temperature conversion, and cycles the LCD display (if the currently selected filter mode is set to ON or OFF to show audio activity). These tasks are created and scheduled by the Salvo™ RTOS Scheduler which uses the MSP430's Timer_A3 as the heartbeat of the system.

The response to the SW1 and SW2 pushbutton presses is handled by a single port Interrupt Service Routine which first decodes which button was pressed and then carries out the corresponding action. SW1 toggles between stop and play operation by sending the appropriate command to the DSP while updating the LCD screen and SW2 cycles through the volume setting numbers and updates the LCD as well.

4 The DSP-Side Reference Framework

The main foundation for the DSP-side application was created directly from TI's eXpressDSP™ Software Reference Framework 3. For the complete details on the RF3 implementation, refer to the application note *Reference Frameworks for eXpressDSP Software: RF3, A Flexible, Multi-Channel, Multi-Algorithm, Static System* ([SPRA793](#)). The following sections describe the software components which were added to the existing RF3 infrastructure after downloading it.

4.1 Additions to the Standard eXpressDSP™ Reference Framework 3

The baseline eXpressDSP™ RF3 application for the C5510DSK was modified to incorporate the extra logic for the RS232 UART daughtercard serial link, as well as the extra layer of code which handles the message passing and processing between the two processors. The subdirectory `\devMSPLink` contains the extra C code that has been added to the standard RF3 baseline release. Since the MSP Link software was written to be a reusable module, it can be easily ported to other projects.

4.2 MSP Link Driver Code (DSP Side)

The MSP Link driver code on the DSP-side is a modular, portable, and reusable software module that should work with any TI DSK and DSPGlobal RS232 daughtercard combination. The DSP simply reads each character which is received by the selected UART channel on the RS232 daughtercard. An interrupt is generated each time a character is received by the UART from the MSP430's on-board USART (configured for UART operation).

When the DSP is ready to process the interrupt generated by the UART, the DSP executes an Interrupt Service Routine to read the contents of the DSP *Inbox* and then creates a copy of the message by placing the data message onto a DSP/BIOS™ QUE (queue) object. A DSP/BIOS™ TSK (task) is responsible for processing the messages pulled off the queue in FIFO order. In this fashion, multiple data messages can be saved and then processed in the correct order by using a single shared memory location for incoming MSP430 messages.

5 Conclusion

The purpose of this application note is to provide the system designer a hardware and software template for creating an application integrating the MSP430 as a host controller with a Digital Signal Processor. Complete software reference frameworks for both the MSP430 microcontroller and TMS320 DSP families have been provided to jump-start system software designs including a method for implementing Inter Processor Communication (IPC) between the two devices.

6 References

1. *MSP430x4xx Family Users Guide* ([SLAU056](#))
2. *MSP430x43x, MSP430x44x Mixed Signal Microcontroller* ([SLAS344](#))
3. *Soft Baugh ES449 Demo Board Users Guide* (www.softbaugh.com)
4. *RS-232-C Serial Communication Daughter Cards Reference Manual* (www.dspglobal.com)
5. *Salvo User Guide & Compiler Reference Manual – IAR MSP430 C* (www.pumpkininc.com)

Appendix A Connecting UART, DSP, and MSP430 Development Boards

This section describes how to set up the hardware platform to run the Hardware UART example which comes with this application note. Both the MSP430 USART and DSP HW UART projects should run without any additional software modifications, provided that all of the hardware connections are set up properly.

1. As per the instructions that come with the DSPGlobal IS-232 UART daughtercard package, mate the daughtercard with the C5510 and install the required jumper on the daughtercard for proper operation specific to the C5510DSK by installing a wire jumper between E7 and E17 on the daughter card. Check with the provided documentation to make sure this connection configuration is correct for C5510DSK operation.
2. Connect the female end of the supplied DB-9 serial cable to Channel 1's DB-9 serial port (marked *DTE Female*) on the RS232 daughtercard. Connect the other (male) end of the cable to the DB-9 serial port on the ES449 board. Ensure that the jumper settings on the ES449 board are configured for normal USART0 operation (consult the ES449 technical reference manual).
3. The two user switches (SW1 and SW2) on the ES449 demo board are not wired to anything (initially out of the box). In order to enable push button operation connect the jumpers J2-1 and J9-1 for SW1 and J2-3 and J10-3 for SW2.

A.1 Setting Up the DSP-side Development Board and Software

Note:

Procedure is based on using Code Composer Studio version 2.20 or above

1. Download the corresponding application note source code provided in the zip file along with this application note and extract the files onto the PC that is used as the host development platform.
2. Connect the audio input source to the DSK board. An easy way to do this would be to connect a 3,5 mm audio cable between your PC's speaker card output and the LINE IN jack of the DSK board. Use a Windows-based application such as Windows Media Player to continuously stream an MP3/WMA/etc. file into the DSK board for processing.
3. Connect the desired stereo output device to the LINEOUT jack of the C5510DSK board.
4. Set up the C5510DSK host-target platform and invoke Code Composer Studio (C5510 DSK CCS) as per the *Quick Start Guide* that comes with the C5510DSK package. Make sure both the DSK and CCS can be started without any communication errors.
5. From the CCS Toolbar, load the provided Workspace file for the sample application [File → Workspace → Load Workspace → **app.wks**].
6. Before proceeding with the next step, be sure to reset the CPU [Debug → Reset CPU].
7. Load the **app.out** [File → Load Program] executable (located in the *\Debug* subdirectory). The absolute path to this file should be `\tms320vc5510 dsk_CCSv220_HWUART \ rf3 \ dsk5510 \ Debug`.
8. Verify that the input sound source is generating an audio stream to the DSK board, then run [Debug → Run] the program. You will not hear any audio output from the DSK until you launch the MSP430-side host application in the next section. At this point, just verify that audio is being fed into the DSK board via its LINE IN jack, and that the DSP is currently running (refer to lower left-hand corner of the CCS GUI and verify that the words *CPU RUNNING* are displayed).

Note:

Each time before loading the application to the target [File → Load Program], be sure to reset the CPU first [Debug → Reset CPU]. Failure to reset the CPU before loading the application executable will result in unpredictable behavior. Do not reset the CPU after loading the application (doing so will wipe out the contents in RAM).

A.2 Setting Up the MSP430-side Development Board and Software

Note:

This procedure is based on the IAR Systems Embedded Workbench 430 V3 IDE.

1. Install the IAR Systems Embedded Workbench 430 IDE. A version can be downloaded from www.ti.com/msp430. If you have not already done so, download the Salvo™ Demo Version and run the installer to put the files into the default of `C:\salvo` (see the Salvo™ O.S. section of this application note for details on installing these files).
2. Using an MSP430-approved JTAG emulator, connect the ES449 demo board to the PC.
3. Launch EW430. From the EW430 IDE Toolbar, select [File → Open Workspace]. Navigate to the subdirectory which contains the MSP430-based project (`\msp430_es449demo_IARv320_USART\src`) within the top-level directory of the downloaded code. Highlight the `msp430f449_rf_iarV3.eww` file and click on the *Open* button. This action will load the provided workspace environment.
4. In the Project window, highlight the `msp430f449_rf_iarV3 – Debug` project name, right-click on it, and then select *Options*. Click on the *C-SPY* Category, and be sure *FET Debugger* is displayed as the Driver. Click the *OK* button.
5. From the EW430 Toolbar, select [Project → Debug]. This step will launch the C-SPY debugger and then erase and program the MSP430 device with the demo program. If any errors are encountered, troubleshoot and resolve the errors before continuing. Select [Debug → Go] to start the program.
6. At this point, the audio input stream to the DSK board should be heard through the speakers. The audio filtering and volume control can be manipulated by pressing the buttons on the ES449 demo board.

Appendix B TMS320VC5510 Software UART Driver via McBSP

In order to save on production Bill Of Material (BOM) costs, the hardware UART used on the DSP side can be eliminated in favor of using one of the available high-speed, full-duplex Multi-Channel Buffered Serial Ports (McBSP) on the DSP itself. A software UART driver can be used to emulate a hardware UART using 3 pins on any of the available DSP McBSP's.

The TMS320VC5510 DSP Starter Kit board has the AIC23 Stereo codec tied to McBSP1 and McBSP2 of the DSP. Thus, McBSP0 can be used to emulate the hardware UART to facilitate the *MSP Link* connection. A sample DSP-side Reference Framework has been provided which includes the Software UART driver configured for use with McBSP0. The complete project is located in the directory **tms320vc5510dsk_CCSv220_SWUART**.

Configuration of the Software UART module is achieved cleanly and easily by modifying the following structure in the file *uartmd_params.c* of the supplied DSP-side project:

```

/* Device params setting for the C5510 DSK software UART driver
 * The following tag needs to be -D defined in the linker build options */
#ifdef defined(_UARTHWDISK5510_MCBSP_)
#define CHIP_5510 1
#include <uarthw_mcbbsp.h>
#include <csl.h>
#include <csl_mcbbsp.h>
#include <csl_dma.h>

/* This structure holds the McBSP parameters and is referenced by the
 * device parameters structure.
 */
UARTHWDISK5510_MCBSP_Params uartMcbbspParams = {
    MCBSP_PORT0,          /* mcbbspId    */
    DMA_CHA1,            /* dmaRxId    */
    DMA_CHA2,            /* dmaTxId    */
    200000000,           /* mcbbspClkIn */
    115200,              /* baud       */
    {
        {
            UARTHWDISK5510_MCBSP_IER_MASK_DEFAULT,
            UARTHWDISK5510_MCBSP_IER_MASK_DEFAULT
        },
        {
            UARTHWDISK5510_MCBSP_IER_MASK_DEFAULT,
            UARTHWDISK5510_MCBSP_IER_MASK_DEFAULT
        }
    }
};

```

Note:

The supplied Software UART DSP project will not run on the hardware setup described in this application note. It should be used as baseline software architecture for a custom target board which utilizes one of the DSP's McBSPs for the Software UART communications link. Additionally, the SW UART driver will not work for baud rates below 57600 bps.

Appendix C Connecting the MSP430 to the C5510 Enhanced HPI (EHPI)

An additional method of connecting an MSP430 to a DSP would be to allow the host controller access to a portion of the DSP's internal and external memories through the DSP's Host Port Interface.

The MSP430 must use its standard GPIO pins to interface to the C5510 Enhanced Host Port Interface (EHPI). This allows the MSP430 to act as an external bootloader to read from and write to certain portions of DSP memory.

In order to connect the two development boards together via the C5510 Enhanced Host Port Interface, a total of 19 wired connections must be made between the two boards. Building a custom ribbon cable is recommended. Using the connectors and wires of your choice, connect the following pins between the two boards as specified in [Table C-1](#). For example, the first row of the table reads connect J2-3 on the ES449 board to P3-13 on the C5510DSK board.

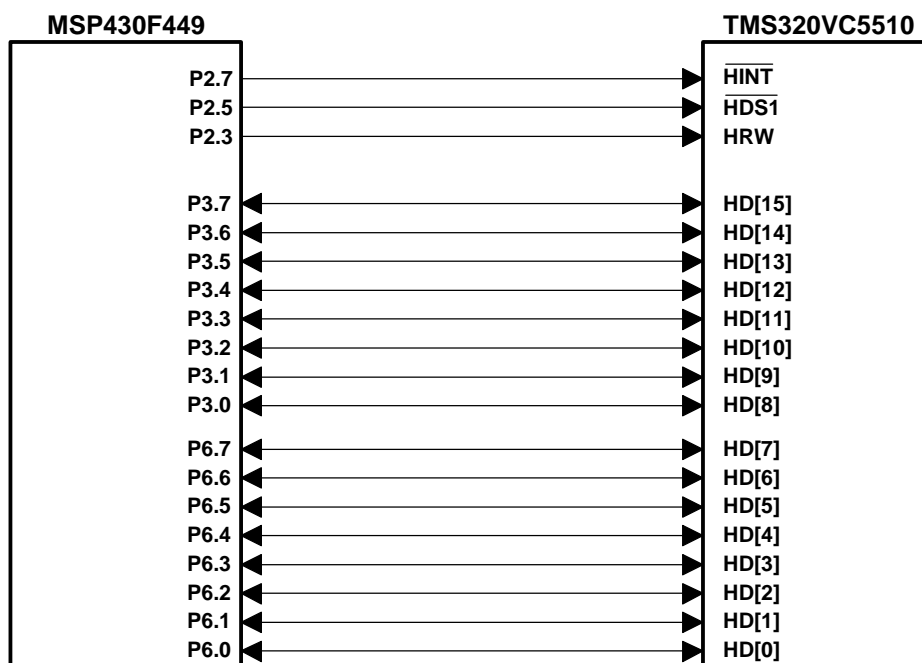


Figure C-1. MSP LINK Using DSP Enhanced Host Port Interface (EHPI)

Table C-1. Pin Connections Between ES449 and DSK5510

ES449			DSK5510		
CONNECTOR	PIN	DESCRIPTION	CONNECTOR	PIN	DESCRIPTION
J2 (Port 2)	4	P2.3	P3 (HPI)	13	HRW
J2 (Port 2)	6	P2.5	P3 (HPI)	20	/HDS1
J2 (Port 2)	8	P2.7	P3 (HPI)	24	/HINT
J3 (Port 3)	8	P3.7	P3 (HPI)	53	HD[15]
J3 (Port 3)	7	P3.6	P3 (HPI)	54	HD[14]
J3 (Port 3)	6	P3.5	P3 (HPI)	55	HD[13]
J3 (Port 3)	5	P3.4	P3 (HPI)	56	HD[12]
J3 (Port 3)	4	P3.3	P3 (HPI)	57	HD[11]
J3 (Port 3)	3	P3.2	P3 (HPI)	58	HD[10]
J3 (Port 3)	2	P3.1	P3 (HPI)	59	HD[9]
J3 (Port 3)	1	P3.0	P3 (HPI)	60	HD[8]
J6 (Port 6)	8	P6.7	P3 (HPI)	63	HD[7]
J6 (Port 6)	7	P6.6	P3 (HPI)	64	HD[6]
J6 (Port 6)	6	P6.5	P3 (HPI)	65	HD[5]
J6 (Port 6)	5	P6.4	P3 (HPI)	66	HD[4]
J6 (Port 6)	4	P6.3	P3 (HPI)	67	HD[3]
J6 (Port 6)	3	P6.2	P3 (HPI)	68	HD[2]
J6 (Port 6)	2	P6.1	P3 (HPI)	69	HD[1]
J6 (Port 6)	1	P6.0	P3 (HPI)	70	HD[0]

The Host Port Interface of the DSP needs to be configured for Multiplexed mode without using the /HAS signal. Tie the following pins to ground.

Table C-2. Pull-Down Resistors for DSK5510 EHPI

CONNECTOR	PIN	HPI FUNCTION	REQUIRED ACTION
P3 (HPI)	15	HMODE (PULLDOWN)	Connect this pin directly to P3-11 (GND) for Multiplexed mode
P3 (HPI)	16	HCS (PULLDOWN)	Connect this pin directly to P3-12 (GND) for constant chip enable

To minimize the number of connections between the MSP430 and DSP, the HPI control signals can be shared with the HPI data lines. The following table describes the remaining connections needed for the HPI pins. Connect the corresponding pins on the C5510DSK.

Table C-3. Internal Connections for DSK5510 HPI in Multiplexed Mode

CONNECTOR	PIN	PIN	REQUIRED ACTION
P3 (HPI)	69 (HD1)	49 (HA1)	Connect these two pins (HD[1] to (HA[1] = HCNTL1))
P3 (HPI)	70 (HD0)	14 (HCNTL0)	Connect these two pins (HD[0] to HCNTL0)

Note:

A sample source code template file for the MSP430 host side (*msplink_dsp55hpi.c*) has been provided to jump-start MSP430 project development; however, caution must be used to follow the exact Host Port Interface timings for the specific C55x-based DSP being used in the design. The provided source code will most likely need to be modified accordingly to meet those timings of the specific DSP being used. The DSP-side project will need to be modified as well so that interprocessor communication is done through EHPI communications rather than UART.

Appendix D Creating a Stand-Alone Embedded System Using FlashBurn™

The FlashBurn™ Utility which comes as an update to Code Composer Studio can be used to program the on-board Flash memory with the application for the C5510DSK. This will allow the application to run without having to connect a host computer to load the program image to RAM each time. In other words, FlashBurn™ will help you to achieve a stand-alone embedded system.

Note:

There is no FlashBurn™-equivalent utility needed for the MSP430.

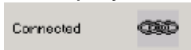
The Code Composer Studio project comes with the necessary files to program an application to the on-board Flash memory on the VC5510DSK board. Follow these easy steps:

1. Turn **off** (uncheck) the *Load Program After Build* option from the CCS Toolbar [Option → Customize → Program Load Options tab].

Note:

Be sure that the CPU is halted and has been previously reset within CCS before attempting the following step. The FlashBurn™ Utility will be unable to establish communications with the board if the CPU is currently running and/or has been halted in an unknown state.

2. Start the FlashBurn™ Utility [Tools → FlashBurn]. If the FlashBurn™ option does not show up in the Tools drop-down menu, the utilities can be downloaded from the Spectrum Digital website (www.spectrumdigital.com) or through the CCS Update Advisor.
3. From the FlashBurn™ Utility GUI, select [File → Load] and select the provided **app.cdd** file. The utility will attempt to communicate with the target board – when successful, the *Connected* message should be displayed in the lower portion of the GUI.



4. Locate and click on the *Browse* button next to the **File to Burn** field. In the mini-browser window that pops up, select the **app.hex** file (by default, it should already be pointing to the project's current working directory).
5. Locate and click on the *Browse* button next to the **FBTC Program File** field. In the mini-browser window that pops up, navigate to your CCS installation directory (most likely C:\ti) and from there, follow the path to the **FBTC55.out** file (most likely \ bin \ utilities \ flashburn \ c5500 \ dsk5510).
6. Save the current changes to the **app.cdd** file [File → Save].
7. Hit the *Erase Entire Flash* icon at the top of the FlashBurn™ toolbar.



8. Hit the *Start Programming* icon located next to the *Erase Entire Flash* button.



9. Remove the connection between the Host PC and the VC5510DSK board recycle power and the application should start up immediately without aid from the Host PC.

D.1 Restoring the Original VC5510DSK Flash Program

By default, each TMS320VC5510DSK is shipped with a self Power-On-Self-Test (POST) program in the on-board Flash memory which runs when power is first applied to the board. To restore the C5510DSK's Flash memory to this original state, repeat the above procedure but with the following replacement file and directory names.

Table D-1. File and Directory Names for Restoring the C5510DSK's POST Flash Program

Working Directory	\apps\rf3\dsk5510	\apps\rf3\dsk5510\post
COFF File (generated by CCS)	<Working Directory>\Debug\app.out	<Working Directory>\Debug\post.out
HEX File (converted from *.out)	<Working Directory>\app.hex	<Working Directory>\post.hex
FlashBurn™ CDD file	<Working Directory>\app.cdd	<Working Directory>\post.cdd

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265