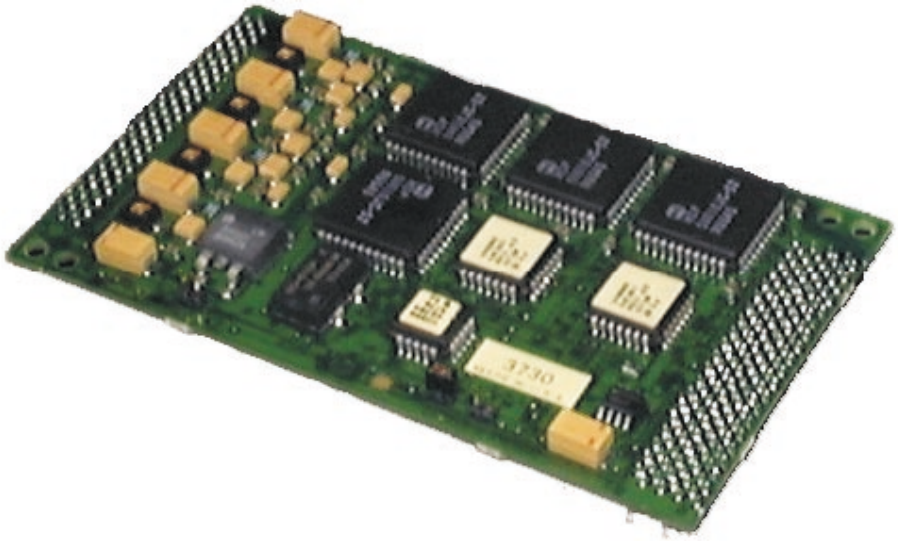
HP SCMVX008 Option 040
4-Channel DDR TIM-40 Board

Programmer's Guide



HEWLETT® PACKARD

Part Number VX008-90010+

Printed in U.S.A.
Print Date: TBD, 1998

8600 Soper Hill Road Everett, Washington 98205-1298 U.S.A.

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure is subject to HP standard commercial license terms or to the following restrictions, whichever is applicable:

- For non-DoD Departments and Agencies of the U.S. Government, as set forth in FAR 52.227-19(c)(1-2)(Jun 1987);
- For the DoD and its Agencies, as set forth in DFARS 252.227-7013 (c) (1) (ii) (Oct 1988), or DFARS 252.221-7015(c) (May 1991), whichever is applicable.

HEWLETT-PACKARD COMPANY
3000 Hanover St.
Palo Alto, CA 94304

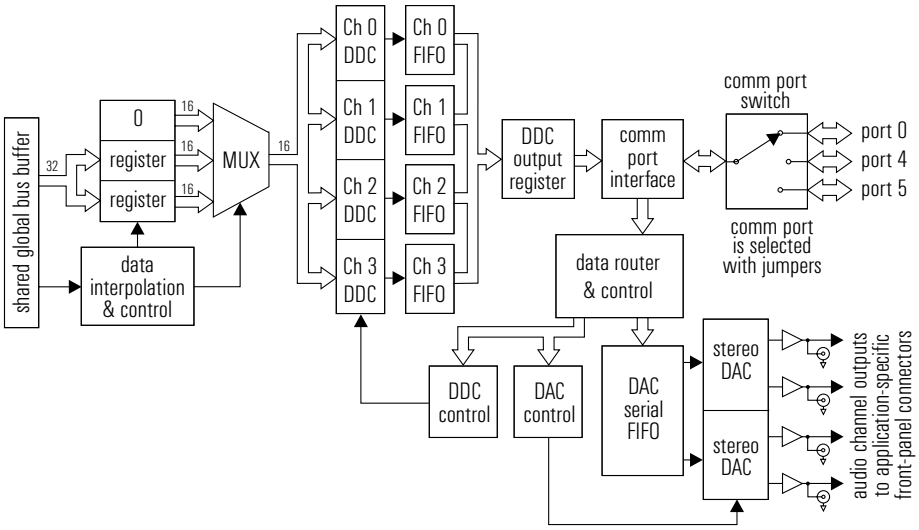
Printing History

Edition 1: January, 1998

Table of Contents

Introduction

The SCMVX008[1] option 040 DDR TIM board is a 4-channel digital downconverter (DDC). It also provides 4 digital-to-analog converters (DAC). See figure below. This TIM-40 module is designed to be installed on the SCMVX008 carrier board (by Spectrum Signal Processing) and is referred to as the DDR TIM elsewhere in this document.



This discussion describes how to install the hardware and software, including a general description of the pieces, where they are located (i.e.. directory names), and how to run the demonstration program.

[1] The SCMVX008 is referred to elsewhere in this document simply as VX8.

Overview

This discussion describes how the DDR TIM and C40 modules may be used to create multi-channel, narrowband receiver systems.

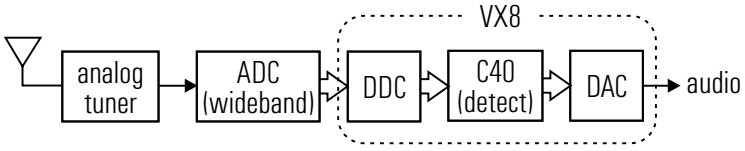
The VX8 Carrier Board has 2 TMS320C40 DSPs permanently installed on it and sites to install 6 TIM-40 plug-in boards. The TIM-40 boards may be either HP DDR TIMs or more C4x DSP TIMs. Default configurations are discussed on page 14.

Digital Drop Receiver

The following figure shows the generic block diagram of a *digital drop receiver* (DDR). Signal data flows through the system as follows:

1. An analog tuner (optional) downconverts the antenna signal to an IF near baseband.
2. A wideband analog-to-digital converter (ADC) digitizes the tuner's IF output.
3. The wideband digital data is transferred over the VXI Local Bus to the VX8 module where it is broadcast to one or more digital downconverter (DDC) modules. The DDC channelizes a signal (tunes to a specific narrowband frequency).
4. The channelized, lower-bandwidth digital signal is transferred to a C4x digital signal processor (DSP) module on the VX8 board via a comm port. The DSP performs detection/demodulation of the data, transforming it to audio information in digital format.
5. The digital audio data is transferred to a digital-to-analog converter (DAC) on the DDR TIM board via comm port and it is converted to analog format.

Generic Digital Drop Receiver



Requirements

The software provided with this product supports Windows NT, Windows 95, and HP-UX 9.0x. There are some differences in the level of support for each as described in more detail later.

Supported system configurations:

- PC embedded slot-0 controller (e.g. HP E6233)
- PC connected via MXI-2 interface (National Instruments VXI-PCI8000 kit) to the VXI chassis
- HP-UX embedded slot-0 controller (e.g. E1498A)
- HP-UX workstation connected via MXI interface (E1482B) to the VXI chassis

Requirements for the `HP Radio` demonstration program:

- Runs on a PC controller, only.
- PC host/controller is Pentium or better
- VXI chassis
- One VX8 VXI module
- At least one DDR TIM installed on the VX8 board.
- For more detail see **Example Program** on page 11.

Terminology

This topic defines the terms, abbreviations, and acronyms used in this document.

Definition of Terms

- *Channelize* – to extract a narrowband signal at a specific center frequency from a wideband digital data stream. One DDR TIM board can *channelize* four signals at any four center frequencies within the ADC bandwidth.
- *Host* – the computer system which starts and controls the high-level processes in a narrowband receiver system; a PC or HP-UX workstation. The HP Radio program runs on the *host*. It downloads the `radio` program to the target system on the VX8 carrier board.
- *Target* – the VX8-based computer system which is started and controlled by the host system in a narrowband receiver system; also referred to as the *dsp* system. It is composed of one or more C4x DSPs on one or more VX8 carrier boards. The function of the target system is to manage the assets and perform the processes of the low-level narrowband receiver system. Commands and responses pass between the host and the *target* via a shared-memory messaging scheme.
- *Embedded* – installed directly in a system chassis or on a VXI module.

  When used to describe a VXI controller, "embedded" means the computer system is, itself, a VXI module installed in slot 0 of a VXI chassis as opposed to an external (desktop) computer which could be used in conjunction with a MXI interface. Embedded controllers have front-panel connectors for a mouse, a keyboard, and a monitor as well as interfaces such as LAN, SCSI, and RS232.

  When used to describe a C40 DSP, "embedded" refers to one of the two TI TMS320C40 DSP ICs installed permanently on the carrier board as opposed to TIM-40 boards which are plugged into node sites on the board. You can get 24 narrowband receiver channels on one VX8 board by installing 6 DDR TIM boards and grouping 3 each with the two *embedded* C40s.
- *Node* – one of the eight locations on the VX8 carrier board which may contain an embedded C40 or TIM-40 module. The nodes are identified by the letters of the alphabet, A through H; nodes A and B contain embedded C40 DSPs and nodes C through H are locations supporting the TIM-40 plug-in modules. See page 4.
- *Comm Port* – TMS320C4x high-speed interprocessor communication feature. The C40 has 6 comm ports and the C44 has 4. See page 6.

Acronyms/Abbreviations

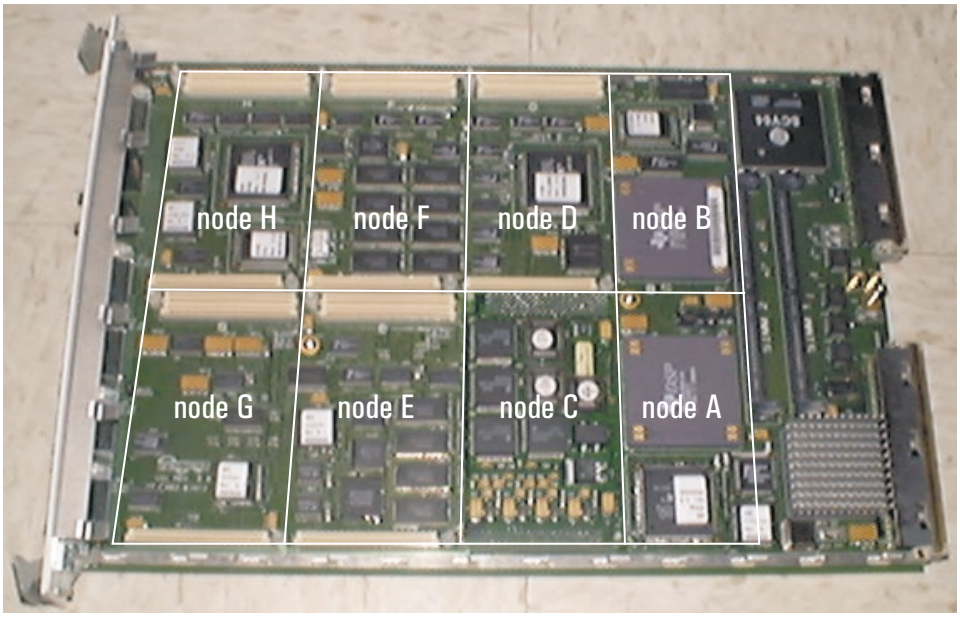
- VX8 – shorthand for Spectrum Signal Processing's SCMVX008 carrier board
- DDC – digital downconverter
- DSP – digital signal processor
- DAC – digital-to-analog converter
- DDR – digital drop receiver (DDC + DSP/detection + DAC/audio out). See **DDR Groups** on page 14.
- TIM – Texas Instruments Module, also used as TIM-40
- C40 – The Texas Instrument TMS320C40 DSP.
- C4x – Refers to either the TI C40 or C44 DSPs. The C40 may be either one of the two embedded ICs installed on the VX8 carrier board or additional TIM-40 boards.
- ISR - interrupt service routine
- EOB - end of block

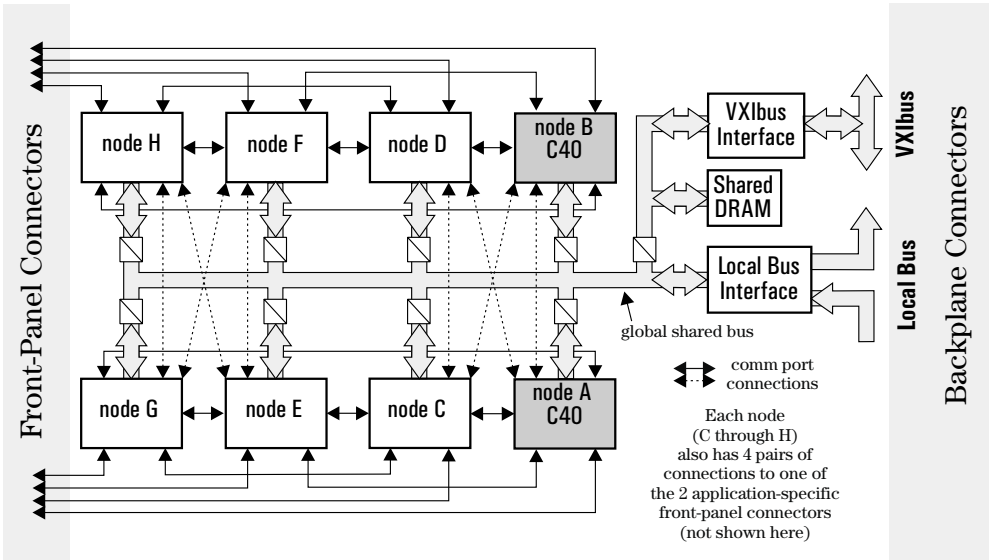
Hardware Installation

Installing TIM-40 Modules

The DDR TIM modules are usually installed on the VX8 carrier board by Hewlett-Packard before they are shipped. They should be configured properly when you receive them. Checking or changing the configuration requires removal of the VX8 carrier board cover. To move, install, or remove a DDR TIM module on the VX8 carrier board, see Spectrum Signal Processing's *VX8 Carrier Board Installation Guide* provided with the VX8 VXI module. See also Appendix A on page 63.

The following figure shows a VX8 with the cover off and one DDR TIM installed in node C (note that the connectors are visible in the other nodes).





Jumpers

The jumpers on the DDR TIM module select the clock source and the module's single comm port as follows. The jumper settings for several example configurations are described in Appendix A: Hardware Configuration on page 63.

Clock Jumper

The clock source may be either the ADCCLK signal from the VXIbus or a clock generated on the VX8 board (SYSCLK). Normally (using the default setting) the ADCCLK signal is selected so that the data processed by the DAC section stays in sync with the ADC that generates the digital information.

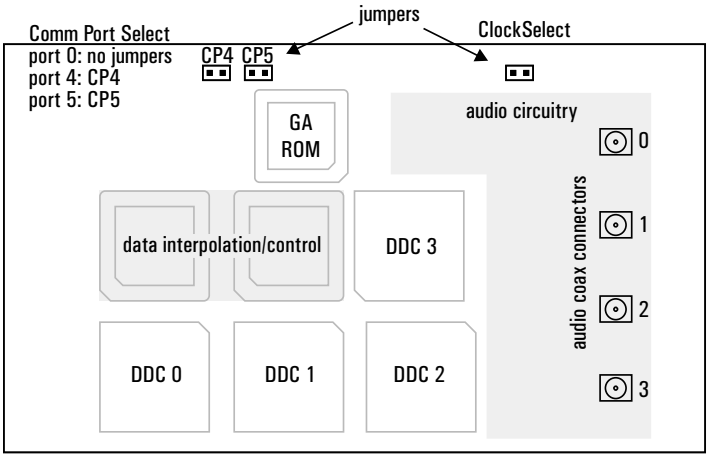
Note

The ADC *must* be set to drive this clock over the ECLTRG1 line by becoming the clock master. DDR programming requires thatl the clock signal is present.

The SYSCLK is provided for the special case in which the DAC section is used and there is no ADCCLK signal; either there is no ADC module to provide it or it is not available on the ECLTRIG1 line of the VXIbus.

The clock selection is made with the Clock Select jumper as follows; see figure:

- **no jumper** selects the ADCCLK signal (default)
- **jumper** selects the SYSCLK signal



Comm Port Jumper

The DDR TIM comm port is selected as follows:

- **Comm Port 0:** no jumper
- **Comm Port 4:** jumper at CP4
- **Comm Port 5:** jumper at CP5

Notes

There is no *default* setting for the comm port jumper. Since they come installed on the VX8 board and must be able to communicate with a C4x DSP, each board's comm port is set at the factory based on its node location and that of its C4x master. See the table on page 6.

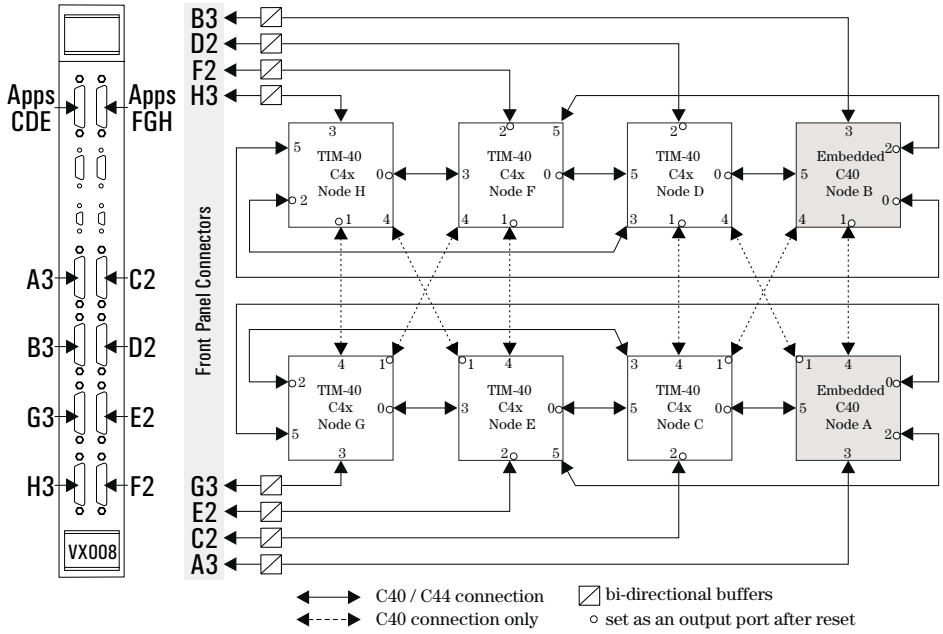
If no jumper is needed, you may place a jumper on one of the two pins to keep it available for future changes.

DDR Comm Port

Comm ports are used to pass data between nodes on the VX8 carrier board as shown in the following figure. The DDR TIM module has a single comm port interface which may be configured as either comm port 0, 4, or 5. The comm port determines which node the DDR TIM module uses to communicate with its C4x controller. The comm port selected depends on the node locations of both the DDR TIM module and the C4x.

The comm port on the DDR TIM is used for several purposes:

1. Channelized, lower-bandwidth data is passed from the DDR TIM to the C4x.
2. Detected/demodulated digital audio is passed from the C4x to the DDR TIM for conversion to analog format by the DACs.
3. Configuration/control information is passed from the C4x to the DDR TIM



Comm Port Selection

The following table lists the comm port to use given the node location of the DDR TIM and the node location of the C4x DSP that will control it and process its data. For more information about group configurations, see **DDR Groups** on page 14.

DDR TIM node	C4x DSP node	DDR Comm Port	DDR TIM node	C4x DSP node	DDR Comm Port	DDR TIM node	C4x DSP node	DDR Comm Port
	A	0		B	0		C	0
C	D*	4	D	A	4	E	F*	4
	E	5		F	5		A	5
	D	0		E	0		F	0
F	G*	4	G	H*	4	H	E*	4
	B	5		A	5		B	5

* A DDR TIM cannot communicate with a C44 module in this position.

Also, the jumper settings for several example configurations are described in Appendix A: Hardware Configuration on page 63.

Audio (DAC) Outputs

The audio outputs of each DDR TIM channel's DAC are routed to the VX8 application-specific connectors as given in the following table:

Node	audio ch. #	Pin # signal	Pin # return	BoB jack*	Node	audio ch. #	Pin # signal	Pin # return	BoB jack*	Node	Audio Ch. #	Pin # signal	Pin # return	BoB jack*
C	0	15	14	1	D	0	19	18	5	E	0	23	22	9
	1	3	4	2		1	7	8	6		1	11	12	10
	2	2	1	3		2	6	5	7		2	10	9	11
	3	16	17	4		3	20	21	8		3	24	25	12
F	0	15	14	1	G	0	19	18	5	H	0	23	22	9
	1	3	4	2		1	7	8	6		1	11	12	10
	2	2	1	3		2	6	5	7		2	10	9	11
	3	16	17	4		3	20	21	8		3	24	25	12

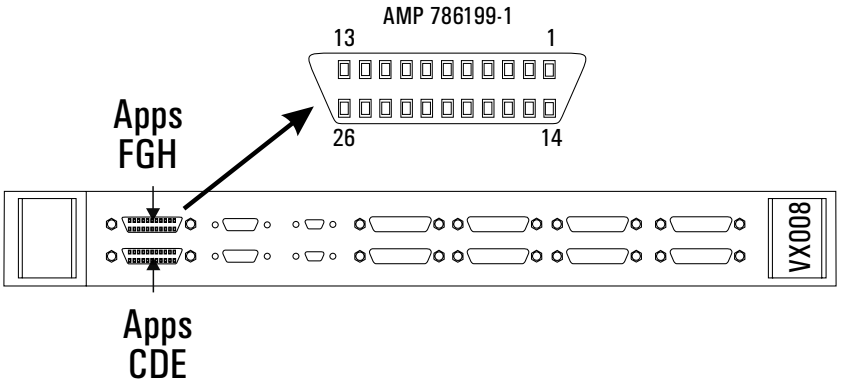
Pin # refers to the pin numbers as shown in the figure directly below.

* **BoB jack** refers to the jack number (1–12) of the audio breakout box (BoB) E3245A shown in the bottom figure. This accessory provides monaural connections.

The following figure shows the pin numbers for the application-specifc connectors. The part numbers for the connector that fit the front panel connectors are:

- Cable connector, plug: AMP 750833-1
- Backshell kit: AMP 750850-3

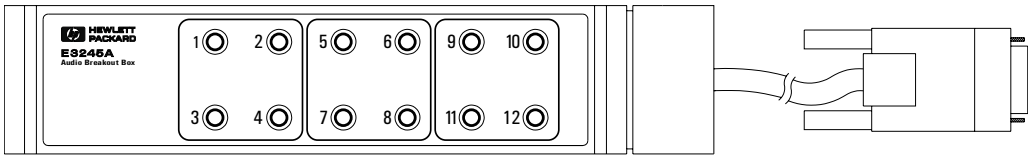
Audio Connections



Note

The DDR TIM board also has 4 coax connectors mounted on the board which may be used to connect to the DAC outputs. See the figure on page 5. Your VX8 carrier board may have a number of small bulkhead connectors on the front panel to which these attach. The connector type is OSMT from M/A-COM. See series part number 9950-4100-xx for available cables.

E3245A Audio Breakout Box Accessory



This accessory provides monaural miniature phone plug connections (3.5 mm as are commonly found on headphones) from the VX8 application-specific front-panel connectors. For connections, see the **BoB jack** column in the table above.

Software Installation

The software provided with the SCMVX008 option 140 software development kit supports C language programming and includes libraries and header files for development in three environments:

- Windows 95
- Windows NT
- HP-UX 9.0x

Windows 95/NT

To install the software on a PC, insert floppy disk #1 into drive A: and run A:\setup. Follow the instructions given by the installation program.

To run the `radio` demonstration program under Windows NT:

- Microsoft service pack 2 or greater should be installed. The installation program checks this and notifies you if the update is not installed. The NT service pack can be obtained from Microsoft's web site, www.microsoft.com.
- VXI plug&play support must be installed. The installation program verifies that `VISA32.DLL` exists in the Windows system directory.
- Spectrum Signal Processing's ssvx8_32.dll must be installed in the `\vxipnp\bin\win`*xx* directory. (*xx* depends on the operating system.)
- See the `readme.txt` file in `\vxipnp\win`*xx*`\hpradio` directory for information about the hardware required to run the HP `radio` demonstration program.

HP-UX 9.0x

To install the software on an HP-UX workstation, insert the DAT (digital audio tape) in the DDS DAT drive and execute the following command as superuser (root):

```
/etc/update -s /dev/update.src "*"
```

This command assumes that the `/dev/update.src` device file refers to the DDS DAT drive. If this device file has not been pre-configured, use SAM (System Administration Manager) to create and configure the appropriate device file for the DDS DAT drive.

Directories and Files

The files and directory names vary depending on the system type.

Windows 95/NT Directories

All installed directories and files are placed in the default directory:

- `c:\vxipnp\winnt` for Windows NT systems
- `c:\vxipnp\win95` for Windows 95 systems

The drive letter may vary but we recommend that the `\vxipnp\winxx` directory is used. VISA library support and drivers for the VXIbus host controller interface should also be in this directory (`\vxipnp\winxx`) as should Spectrum Signal Processing's `ssvx8` host libraries for the SCMVX008 Carrier Board.

The following directories and files are installed for Window 95/NT systems:

- **`include\`** – library and VXI plug&play include files
- **`hpvx8.h`** – the main hpvx8 library include file
- **`lib\`** – archive libraries
- **`msc\`** – the Microsoft-C compatible libraries
- **`hpvx8.lib`** – the hpvx8 static library
- **`hpvx8\`** – top directory for the HP libraries and example programs
- **`readme.txt`** – describes files and directories installed
- **`ddrifc\`** – contains **d**igital **d**rop **r**eceiver **i**nter**f**a**c**e code
- **`readme.txt`** – describes the ddrifc code
- **`dsp\`** – ddrifc target C code files
- **`include\`** – contains ddrifc target include files
- **`src\`** – contains the ddrifc target source files
- **`hpvx8\`** – contains the hpvx8 code
- **`readme.txt`** – describes hpvx8 code
- **`dsp\`** – hpvx8 target C code
- **`include\`** – hpvx8 target include files
- **`src\`** – hpvx8 target source files
- **`host\`** – hpvx8 host C code
- **`build\`** – hpvx8 host build and library files
- **`include\`** – hpvx8 host include files
- **`src\`** – hpvx8 host source files
- **`radio\`** – host & target C programs for the example program
  (more detail for this directory appears on page 50)
- **`dsp\`** – radio target C code
- **`build\`** – radio target build files
- **`include\`** – radio target include files
- **`src\`** – radio target source files
- **`host\`** – radio host C code
- **`include\`** – radio host include files
- **`hpradio\`** – GUI directory for the `radio` example code
- **`readme.txt`** – information about the HP Radio example code
- **`resource\`** – radio resource files

HP-UX Directories

All directories and files are installed in `/opt/vxipnp`. SICL/VISA library support and drivers for the VXIbus host controller interface should be placed in the same directory (`/opt/vxipnp`) as should Spectrum Signal Processing's `ssvx8` host libraries for the SCMVX008 Carrier Board.

The following directories and files are installed for HP-UX systems:

- **`bin/`** – shared libraries
- **`libhpvx8.sl`** – the hpvx8 shared library
- **`include/`** – contains library and VXI plug&play include files
- **`hpvx8.h`** – the main hpvx8 library include file
- **`lib/`** – archive libraries
- **`libhpvx8.a`** – the hpvx8 static (archive) library
- **`hpvx8/`** – top directory for the HP libraries and example programs
- **`readme.txt`** – describes files and directories installed
- **`ddrifc/`** – contains **d**igital **d**rop **r**eceiver **i**nter**f**a**c**e code
- **`readme.txt`** – describes the ddrifc code
- **`dsp/`** – ddrifc target C code files
- **`include/`** – ddrifc target include files
- **`src/`** – the ddrifc target source files
- **`hpvx8/`** – the hpvx8 code
- **`readme.txt`** – describes hpvx8 code
- **`dsp/`** – hpvx8 target C code
- **`include/`** – hpvx8 target include files
- **`src/`** – hpvx8 target source files
- **`host/`** – hpvx8 host C code
- **`build/`** – hpvx8 host build and library files
- **`include/`** – hpvx8 host include files
- **`src/`** – hpvx8 host source files
- **`radio/`** – host & target C programs for the example program
  (more detail for this directory appears on page 50)
- **`readme.pdf`** – (this file) includes details about the `radio` example code
- **`dsp/`** – radio target C code
- **`build/`** – radio target build files
- **`include/`** – radio target include files
- **`src/`** – radio target source files
- **`host/`** – radio host C code
- **`include/`** – radio host include files
- **`hpradio/`** – GUI directory for the `radio` example code
- **`readme.txt`** – information about the HP Radio example code
- **`resource/`** – radio resource files

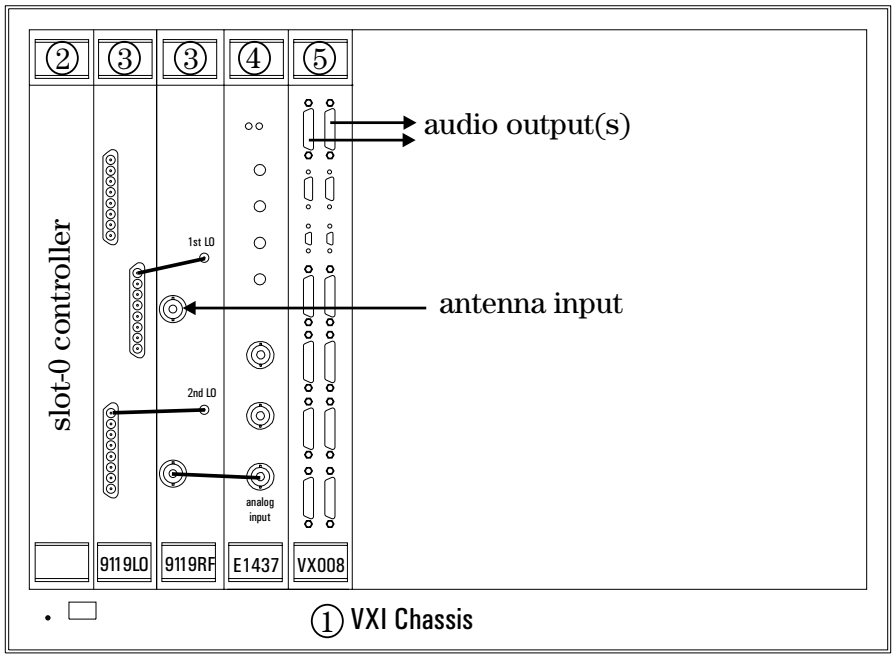
Example Program

This discussion describes the hardware and software required to run the example program `HP Radio`. More information is available in the `readme.txt` file in the `hpradio` directory.

`HP Radio` uses the hpvx8 library to create a multi-channel, narrowband receiver system. It provides a graphic user interface (GUI) which allows you to exercise the capabilities of the narrowband receiver system hardware.

Hardware Configuration

Before you can run the program, you must collect the required hardware and software and configure them correctly. The following figure shows an example hardware configuration.



Installed Hardware

The numbers in the following list correspond to circled numbers in the figure above:

1. VXI chassis/mainframe
2. PC slot-0 controller (NI MXI-2 interface to a PC workstation)
3. Analog downconverter (optional; the WJ-9119 is an HF tuner; 100 kHz-32 MHz)
4. Wideband ADC (the E1437A has an 8 MHz BW)
5. VX008 is a VXI carrier board containing 2 embedded C40s and 6 sites (nodes) that accept TIM-40 plug-in modules such as the HP DDR TIM and the MDC40SS Super SRAM C40
   - At least 4 MB DRAM installed on the VX008
   - At least one DDR TIM installed on the VX008

Note

The model type and configuration of the analog tuner and wideband ADC are declared/defined for the example program in the file `hpradio.cfg`.
See **Resource Files** on page 12.

VX8 Configuration

The system and VX8 hardware configuration must be defined in resource files that the software uses to define the elements it controls.

Resource Files There are four resource files required by the radio program. These files define the hardware configuration and are used by the libraries to initialize the system. They must match the hardware configuration used with radio.

- `hpradio.cfg` – system hardware configuration file
- `hpradio.grp` – hpvx8 group resource file
- `hpradio.sdf` – ssvx8 system definition file (for VX8 configuration)
- `hpradio.ldf` – ssvx8 load definition file (for VX8 configuration)

These files are installed in the `\vxipnp\win`*xx*`\hpvx8\hpradio` directory. The resource subdirectory contains templates and examples for five different VX8 hardware configurations. The resource files listed above define a configuration consisting of one group composed of one C40 and one DDR TIM. The templates support 1x1, 2x1, 2x2, 2x3, and 4x1 configurations where, given *n*x*m*, *n* is the number of groups and *m* is the number of DDR TIMs per group. For example, to change resource files for a VX8 with DDR TIMs in nodes C and D, use the following command:

```
copy resource\2x1.* hpradio.*
```

If the files provided do not match your hardware exactly, select the closest configuration and edit the files as explained in the `readme.txt` file in the `hpradio` directory.

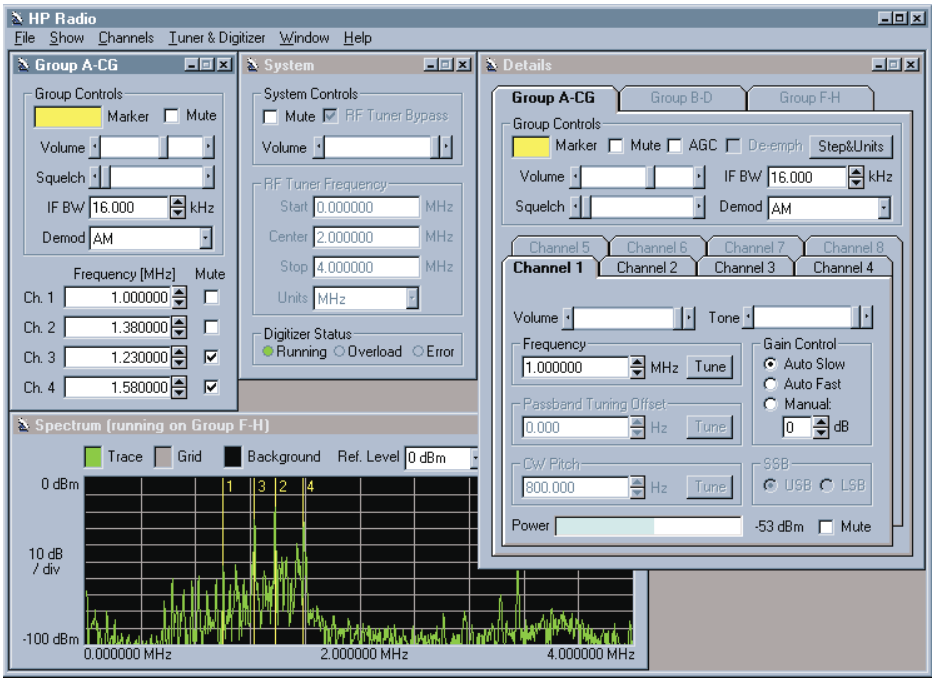
For more information about the various supported receiver configurations, see **DDR Groups** on page 14.

Running HP Radio

When the hardware and resource files are properly configured, you can run the example program.

To start it, either:

- Click on the HP Radio menu item in the Start\Programs\HP VX8 Radio\ menu or
- In Explorer, double-click Hpradio in `\vxipnp\win`*xx*`\hpvx8\hpradio`



The first time HP Radio is run, the interface shows one group window with two channels active. This group window is labeled Group A-C, which refers to a group consisting of a C40 in node A and a DDR TIM slave at node C. Node locations are shown in the figure on page 6.

With one DDR TIM installed, 4 channels are present in this group. In this case, the number of channels that may selected from the Channels menu is either 1, 2, or 4. You may want to select a number less than 4, depending on the bandwidth and modulation type of the signals you plan to receive. Since all four channels are being processed by the same DSP, more channels means less processing power per channel. The maximum real-time IF bandwidth (IF BW) decreases when the number of active channels is increased.

To set the IF BW, either increment or decrement it with the spin buttons or enter a value in the text box. The amber User LED on the VX8 front panel will illuminate when the processing workload has become greater than the processor's ability to handle it *real time*. It will continue to process data but there will be gaps in it. See the discussion on page 47.

Note

More information exists concerning the operation of HP Radio in the readme.txt file in the hpradio directory.

DDR Groups

One, two, or three[2] DDR TIMs may be connected to a single C40 DSP[3] on the VX8 to form a *group*. A group is composed of one C4x (DSP) and the DDR-TIM(s) associated with it. The IF bandwidth is the same for all channels in a group. The DDC channels may be individually tuned to any frequency in the bandwidth of the wideband digital data stream created by the ADC. See **Grouping** on page 23.

The following figures are block diagrams showing data flow through the receiver system for various configurations. The group configuration for the example program HP Radio is defined in the file `hpradio.grp`. See **Resource Files** on page 12.

Example Configurations

The default configurations available from Hewlett-Packard are:

- 1×1 – 1 group, 1 DDR TIM per group; 4 channels per VX8.
- 2×1 – 2 groups, 1 DDR TIM per group; 8 channels per VX8.
- 2×2 – 2 groups, 2 DDR TIMs per group; 16 channels per VX8.
- 2×3 – 2 groups, 3 DDR TIMs per group; 24 channels per VX8.
- 4×1 – 4 groups, 1 DDR TIM per group; 16 channels per VX8.

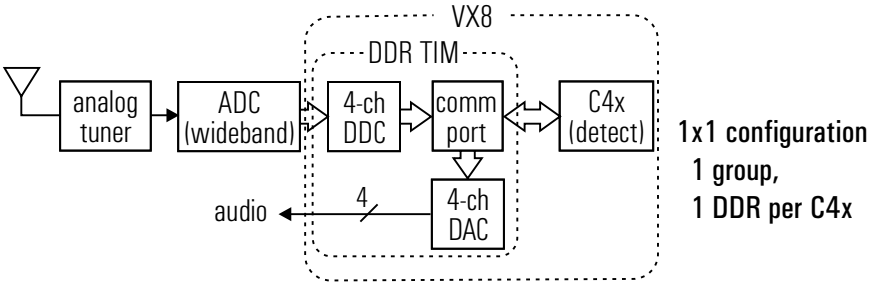
These configurations offer various numbers of channels per VX8 module and varying amounts of digital signal processing power per channel.

[2] Four DDR TIMs per C40 is also possible if you install them in nodes C, D, E, and G with a corresponding reduction in available processing power per DDR. See the figure on page 6.[222222]

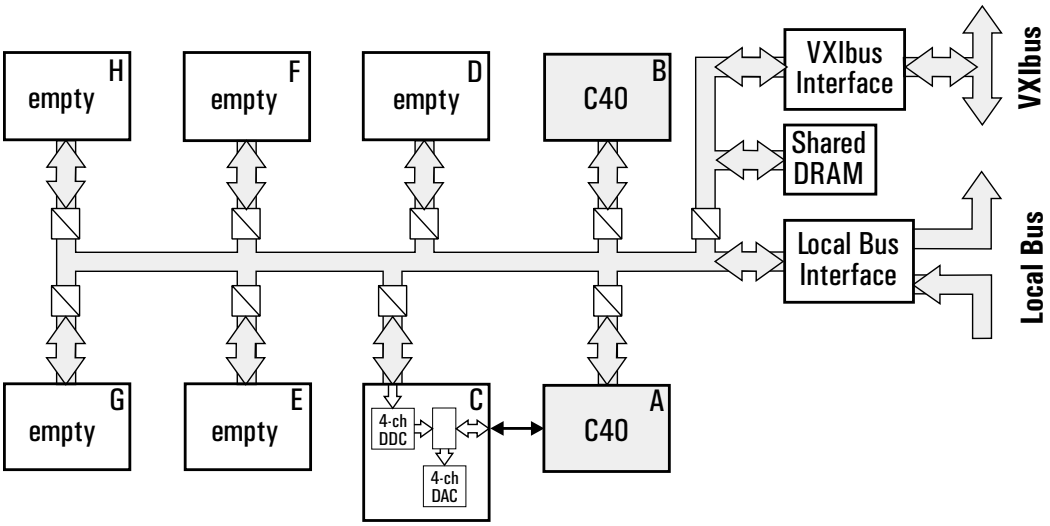
[3] The number of channels that can be processed simultaneously by one C40 DSP is limited by the bandwidth and detection type. See the table on page 60.[333333]

4-channel, 4ch/DSP

The configuration shown below could be created by installing a DDR TIM at node C and grouping it with the embedded C40 at node A. See bottom figure.



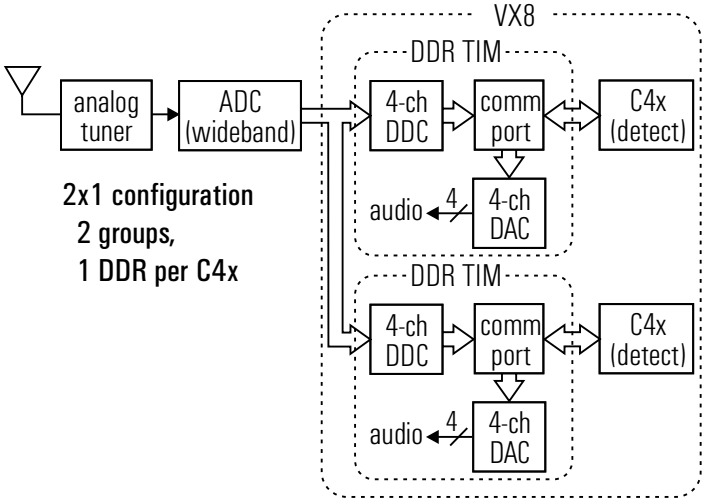
This configuration provides 4 high-performance receiver channels.
The VX8 configuration layout is illustrated in the figure below.



Detailed installation information for this configuration exists on page 64.

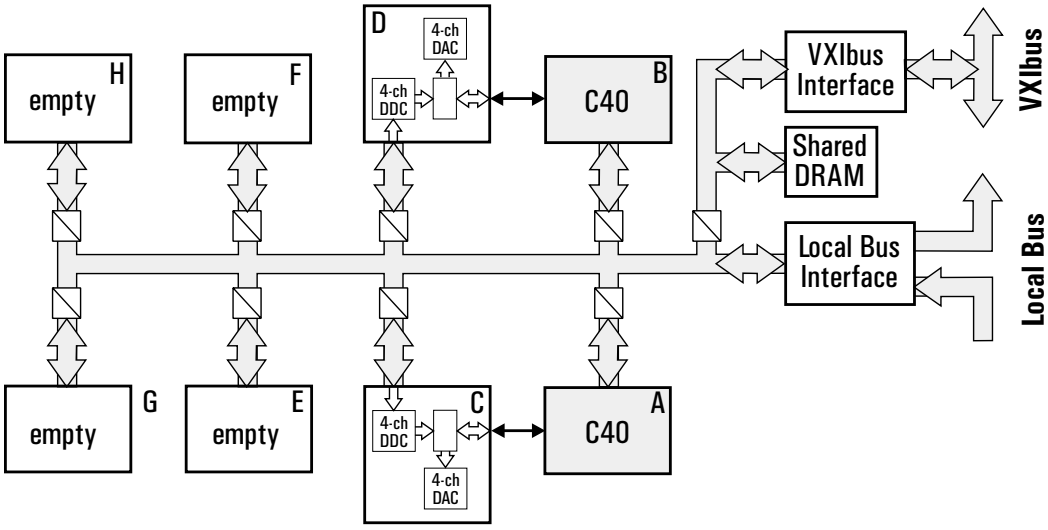
8-channel, 4ch/DSP

The configuration shown in the figure below could be created by installing DDR TIMs at nodes C and D (see bottom figure), grouping them with the embedded C40s at nodes A and B, respectively. These groups are identified as A-C and B-D.



This configuration has virtually the same performance as the 1x1 configuration (4 ch/DSP) with double the number of channels (8) and the ability to process 2 different bandwidths simultaneously.

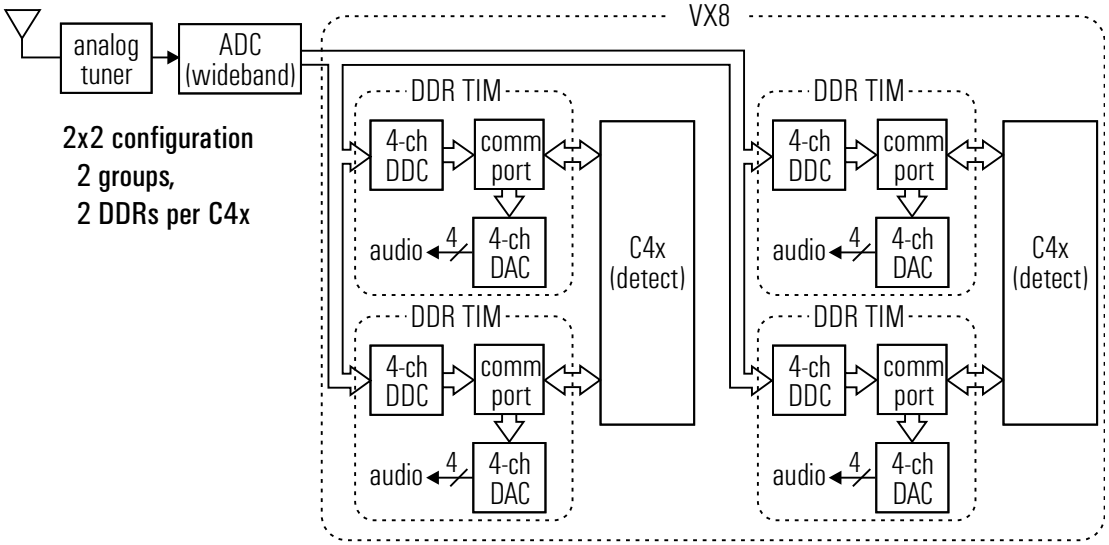
The VX8 configuration layout is illustrated in the figure below.



Detailed installation information for this configuration exists on page 65.

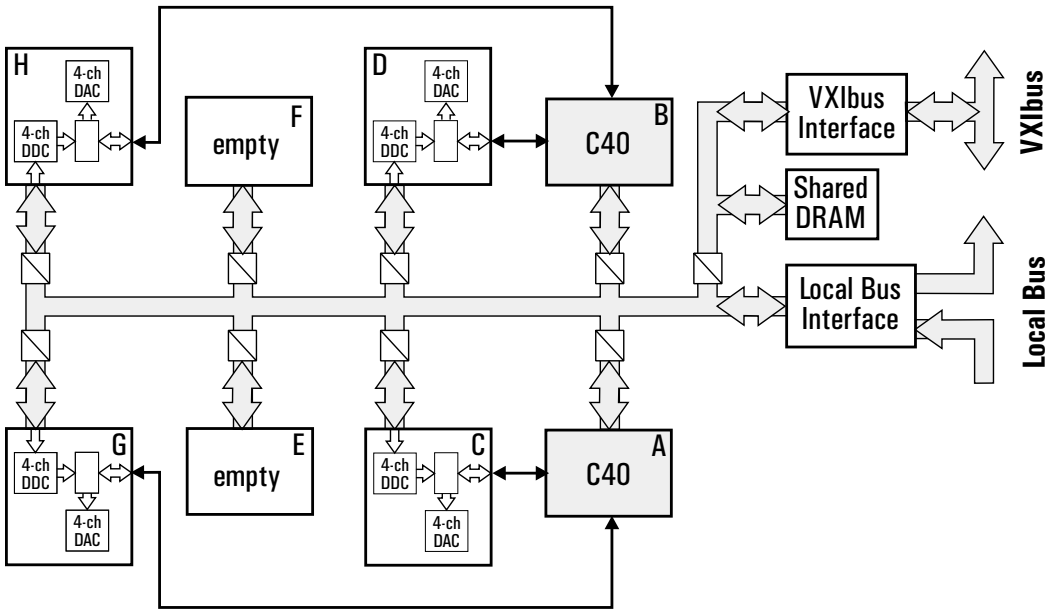
16-channel, 8ch/DSP

The configuration shown in the figure below could be created by installing DDR TIMs at nodes C, D, G and H (see bottom figure), grouping them with the embedded C40s at nodes A and B, respectively. These groups are identified as A-CG and B-DH.



This configuration provides 16 channels and 2 different simultaneous bandwidth settings. Its performance and cost/channel falls between the 2x3 and 4x1 configurations.

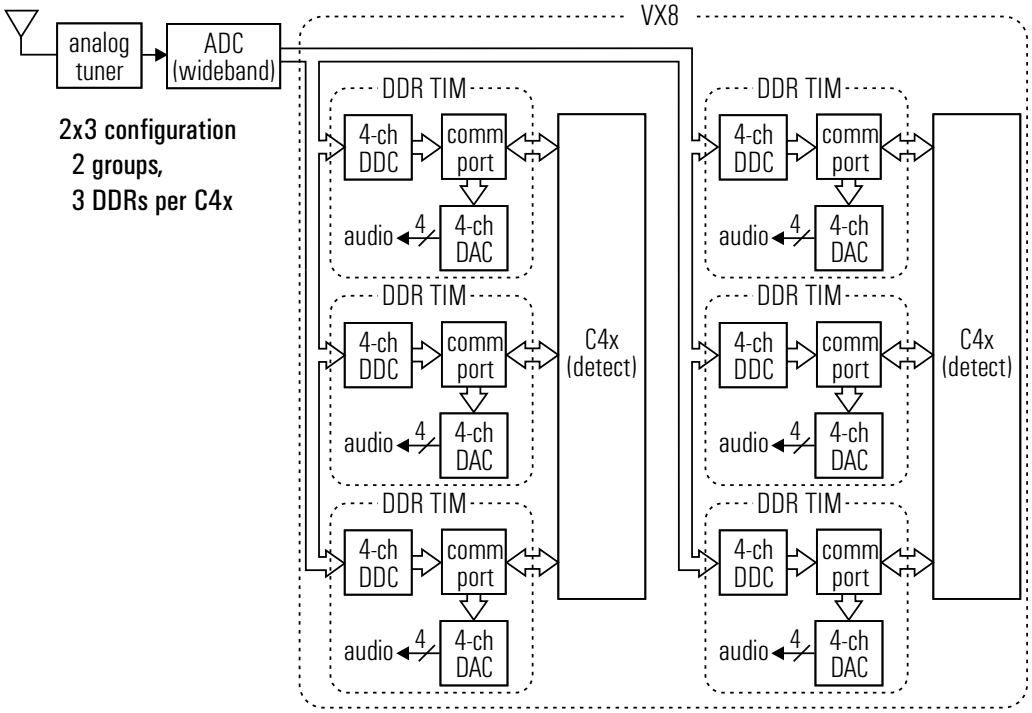
The VX8 configuration layout is illustrated in the figure below.



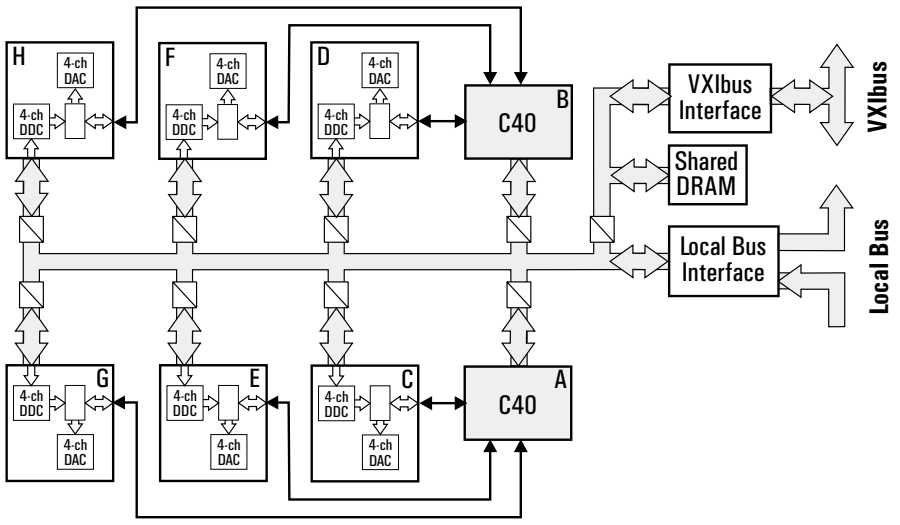
Detailed installation information for this configuration exists on page 66.

24-channel, 12ch/DSP

The configuration shown in the figure below could be created by installing DDR TIMs at nodes C, D, E, F, G, and H, grouped with the embedded C40s at nodes A and B, respectively. These groups are identified as A-CEG and B-DFH.



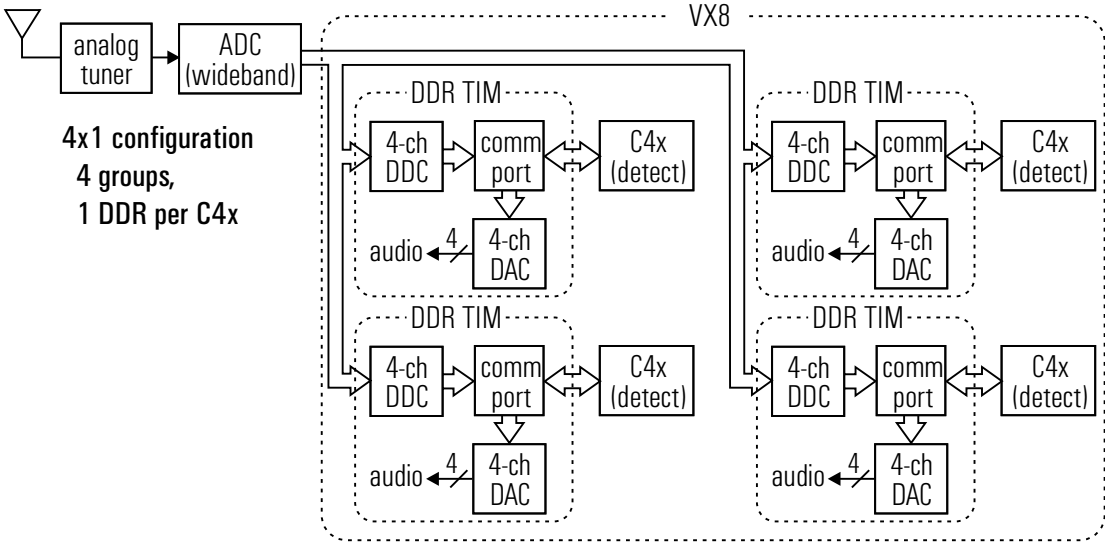
This configuration offers the highest channel-count per VX8 board (24) but also has the lowest DSP performance per channel. Possible applications include use for very narrow-bandwidth signals or for channelization-only, where the demodulation is performed on another system or VX8 board. In the latter case, the C40s would be used to multiplex the channel data onto one or two comm ports and pass it to connectors on the front panel where they are passed to the other system/board.



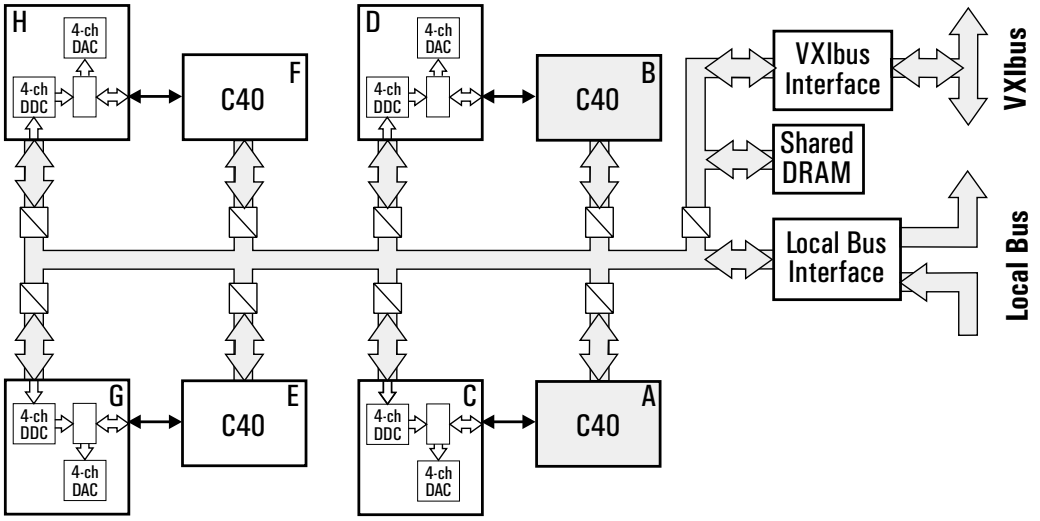
Detailed installation information for this configuration exists on page 67.

16-channel, 4ch/DSP

The configuration shown in the figure below could be created by adding C40 TIM boards at nodes E and F and DDR TIMs at nodes C, D, G, and H. These can be grouped with one DDR per C40 and identified as groups A-C, B-D, E-G, and F-H.



This configuration offers the highest performance (DSP/channel) and most flexibility (4 different bandwidths processed simultaneously). The VX8 board would be loaded as shown in the figure below.



Detailed installation information for this configuration exists on page 68.

System Software Development

Overview

Software development for HP's DDR TIM module involves the following tasks:

- Development of software that runs on the host computer (VXI slot 0 controller) that communicates with and controls the VX8 and other VXI modules. This software is referred to as the *host program*.
- Development of software that runs in the target DSP (VX8 C4x DSP) that controls local resources on the VX8 (e.g. HP DDR TIM) and other resources within the VXI chassis (e.g. ADC). This software is referred to as the *target program*.
- Development of a command, control, data, and synchronization interface between the host and target programs to coordinate overall system operation.

This is an iterative development process whereby coordinated changes in the host and target software must be made. This allows incremental progress in the overall system development while the host and target programs continue to work in concert with one another. Host and target software examples and target source code are supplied to help enable the learning process.

Software Requirements

Software development for a VXI system containing HP's DDR TIM module requires a working knowledge of the following subject areas:

- "C" programming language. "C" is the primary software development language used in both the host and target development environments. Even though it is not required, it is recommended that the target software developer also have a working knowledge of the TMS320C4x assembly language.
- VXI architecture. For more about VXI, please refer to *Feeling Comfortable with VXI*, HP PN: 5965-6497E. For detailed insight, see IEEE Std. 1155-1992 *IEEE Standard for VMEbus Extensions for Instrumentation: VXIbus* or IEEE Std. 1014-1987, *IEEE Standard for a Versatile Backplane Bus: VMEbus*.
- VX8 DSP VXI module architecture. For detailed insight into the VX8, please refer to both the Technical Reference Manual and the Programming Guide supplied with the VX8 from Spectrum Signal Processing.
- TMS320C4x DSP architecture. For detailed insight into the TMS320C4x architecture, please refer to the TMS320C4x User's Guide supplied with the VX8 from Texas Instruments.
- Host and target software development environments. This includes compilers, linkers, debuggers and build tools (e.g. "make"). For host development, one of the following environments is recommended:
  - HP-UX Workstations: "ANSI-C Development Bundle"
  - PC Workstations: Microsoft Visual C++ (version 5.0 or later)

For target development, you need Texas Instrument's TMS320C4x Code Generation Tools, Revision 4.70. It is recommended that you have the PC Release of these tools. In addition to the basic Texas Instrument's code generation toolset, it is also

recommended that a third-party debugger/development toolset be employed for target development, e.g. GO DSP's Code Composer.

Hardware Requirements

Software development for a VXI system containing HP's DDR TIM module requires the following hardware:

- VXI chassis with slot 0 controller. The slot 0 controller can be either an HP-UX or PC workstation. HP-UX support includes both embedded (E1498A) and external (via E1482B MXI) controllers. PC support includes both embedded (E623xA) and external (via National Instrument's VXI-PCI8000) controllers.

  Once the appropriate controller has been chosen, you must install and configure the appropriate I/O layer libraries to communicate with the VXI backplane. Choices of I/O layers include either SICL (Standard Instrument Control Language) or VISA (Virtual Instrument System Architecture). Refer to the documentation for your particular slot 0 controller for more details.

- A VX8 Carrier module with shared DRAM SIMM and one or more HP DDR TIM modules installed. For details on installing the shared DRAM SIMM, please refer to Spectrum's VX8 Carrier Board Installation Guide. For details on installing the HP DDR TIM, please refer Hardware Installation on page 4.

- An external PC with XDS debugger hardware. This external PC is the target software development workstation. This PC should have the TI TMS320C4x code generation tools loaded as well as any third-party development tools (described in Software Requirements section). This PC interfaces with the VX8 via the XDS debugger hardware through the VX8's front panel JTAG connector.

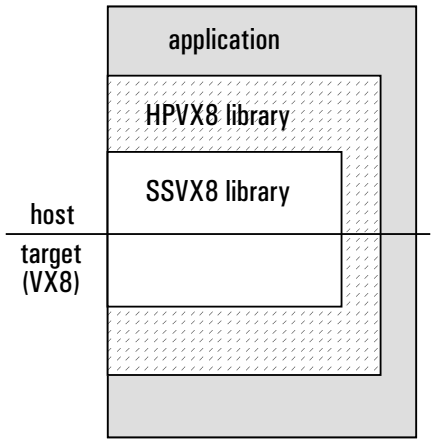
  The XDS debugger hardware consists of a plug-in PC card with a ribbon-cabled pod. This ribbon-cabled pod coming from the PC connects to a small JTAG conversion board that provides a connection to the JTAG connector on the VX8 front panel.

  The XDS debugger hardware is available from either Texas Instruments (XDS-510) or Spectrum Signal Processing (XDSC40). The JTAG conversion board and VX8 JTAG cable is available as a development option to the VX8.

HPVX8 Library

The hpvx8 library provides the means to quickly develop multi-channel, narrowband receiver systems based on the VX8 VXI carrier board, the HP DDR TIM module, and C40/C44 TIM modules. It provides control of the narrowband DDR channelizer modules as well as a method for a host program to communicate with a target program.

The hpvx8 library is built on top of Spectrum Signal Processing's ssvx8 host and DSP (target) libraries. The following figure illustrates how program layers in the host are used to communicate with their counterparts running on the target.



The hpvx8 communication strategy uses a FIFO-based, shared-memory method which also supports communication between DSP units on the VX8 carrier board. This method is one of many possible communication schemes; the source code is provided to allow the developer to explore this approach. The radio demonstration program is an example of how to use the library.

Further information concerning contents of individual files, messaging schemes, and build instructions resides in the `readme.txt` file in the `hpvx8` directory.

Program Development

Grouping

The hpvx8 library requires a group resource file (.grp) to organize C4x DSPs and DDR TIM boards into groups as well as providing some VX8 board setup information. Examples of the .grp file are in the hpvx8/hpradio directory (hpradio.grp) and the hpvx8/hpradio/resource directory (6 *.grp files). These .grp files provide support for various hardware configurations and must match their corresponding .sdf files (see same directories).

The hpradio.grp file is used by the hpvx8 library to determine the groupings of C4x DSPs and DDR TIM modules. The concept of a group is used to describe a collection of processing and DDR components. For HP Radio, a group consists of 1 C4x DSP and 1, 2, or 3 DDR TIM modules. In general, systems built with the hpvx8 library define a group to be a C4x master and a number of C4x and DDR TIM slaves (limited by the number of communication ports). The C4x slaves can in turn have a number of C4x and DDR slaves. The HP Radio example restricts a group to one C4x master with only DDR modules as slaves.

The hpradio.grp file also specifies the number of VX8 boards in a system and some parameters specific to VX8s with DDR TIM modules. In general, the hpvx8 library allows any number of VX8 boards to specified as long as they are all in the same VXIbus chassis. However, HP Radio is programmed to use only one board.

The `hpradio.grp` file also defines the following DDR parameters:

- the HP local bus mode
- HP local bus DMA wait states and target values
- mapping the IRQ connections.

In general, the .grp files provided in the resource subdirectory should either match or closely approximate the configuration of your VX8 hardware. Comments within the .grp file are provided to help you edit the file according to VX8 configuration.

Note

If using the HP E1430A digitizer, the DMA wait states parameter should be set to '5'. If using the HP E1437A digitizer, the DMA wait states should be set to '1'.

One digitizer should be selected as the clock (F_s) master on the VXI backplane (ECLTRG1). Other digitizers and VX8s receive this clock. The DDR TIM cards should have their clock selection jumper removed to use this clock.

System Definition

The system definition file (.sdf) is required by the ssvx8 library to specify the logical address used by the VX8 and the TIM modules installed. The group resource (.grp) file must match the hardware configuration described in the .sdf file. When the .grp file references an empty site, an error occurs. Note that the .grp file contains the names of the .sdf and .ldf that are used by the program accessing the .grp file. The .ldf (load definition file) is also used by the ssvx8 library to determine which target .out file gets loaded to each DSP. See page 12.

See Spectrum Signal Processing's *VX8 Carrier Board Programming Guide* for more information on the ssvx8 library and the .sdf and .ldf files. See the other readme.txt files distributed with this software for further system information.

The hpvx8 library source files that are involved in extracting information from the .grp and .sdf files are hpvx8res.h, hpvx8res.c, hpvx8sdf.h, and hpvx8sdf.c. The grouping concepts are used throughout the library and are defined and manipulated in hpvx8ifc.h and hpvx8ifc.c. Refer to these source files for more information.

Host↔Target and Target↔Target Messaging

The hpvx8 library builds a communication system between the host and target groups and between target master nodes and their slaves. Recall that each group has one master C4x DSP. The communication takes place between the host and the master of a group and between a master node and its slaves. Any messages that need to be communicated to a slave node are first passed through the group master.

For host↔target messaging, a command and return-messaging FIFO structure is created in the VX8 shared DRAM. This requires that the VX8 have at least one 4 MB DRAM SIMM installed. The location of the messaging FIFOs are set at startup (see hpvx8_system_open() below). The hpvx8 library source files involved with host↔target communication are hpvx8ifc.h and hpvx8ifc.c on the host side and hpvx8msg.h and hpvx8msg.c on the DSP (or target) side. Refer to these source files for more information.

For target↔target messaging the same type of messaging FIFO structure is created in each of the slave's near-global SRAM. The hpvx8 library source files that are involved with target↔target communication are hpvx8msg.h and hpvx8msg.c. Refer to these source files for more information. The HP Radio example does not make use of hpvx8 library target↔target messaging.

Host↔target Commands

A set of basic commands (and replies) are defined as a starting point for host↔target communication. Normally, an application requires a set of unique commands and replies which can easily be added to the basic set as long as the formats are kept the same. The list of defined commands and replies are as follows:

`HPVX8_CMD_AdcSetup`

The group master that receives this command is responsible for monitoring and throttling the ADC.

`HPVX8_CMD_DdcSetup`

The five-bit tag passed with this command is used to identify a DDR module. This information is encoded into each sample point coming out of the DDR module.

`HPVX8_CMD_AssignStat`

This command assigns a channel statically, meaning it remains assigned until reassigned. The argument specifies a tuned frequency for the channel.

`HPVX8_CMD_AssignDyn`

This command assigns a channel dynamically, meaning it is assigned until some specified signal-related condition causes it to be released. When the channel is released, the target returns a _Released message to the host (see below).

HPVX8_CMD_Start

This command starts the group, or moves the group from the configuration state to the process state. If a group did not receive an _AdcSetup command, the HP local bus DMA is set up and waits for data. If a group did receive an _AdcSetup command, it querys the ADC for block available before enabling the HP local bus DMA.

HPVX8_CMD_Stop

This command stops the group, or moves the group from the process state back into the configuration state.

HPVX8_CMD_Ifbw

This command sets the IF bandwidth for all DDR TIM modules in the group.

HPVX8_RET_CmdReply

This is a reply to any of the above commands. The target must issue a reply to the above commands. This is not a requirement for user-defined commands.

HPVX8_RET_Released

This reply is used to notify the host that a channel has been released due to a pre-specified condition of the data stream. It would occur only after a channel has been Assigned Dynamically (see above).

HPVX8_RET_Error

This message is used to notify the host that an error has occurred.

HPVX8_RET_AdcError

This message notifies the host that an ADC error has occurred. The error code is generated by ANDing the ADC status register contents and the error mask that is provided with the AdcSetup command. Note that this message can be sent only by nodes which have received the AdcSetup command.

See the file hpvx8cmd.h for more details on the format and description of these commands.

Host Functions

The following functions are included in the hpvx8 host library (see `hpvx8ifc.c`).

`hpvx8_system_open(STRING rsrc_file, HPVX8_SYS *sys, FLOAT64 time_out)`

Arguments: **`rsrc_file`** – name of the group resource file (.grp)
`sys` – pointer to hpvx8 system to be created
`time_out` – time out (in seconds) parameter for all target accesses

Returns: Status

Synopsis: Opens a VX8 system and loads code to target DSPs.

1. First this function reads and parses the group resource file (.grp) to obtain information about the grouping of DSPs and DDRs
2. Then the group resource information is checked for validity.
3. Next, a call is made to ssVX8_SystemOpen() to open the VX8 system, reset it, and load code to the target DSPs. This call reads and parses the .sdf and .ldf files for information about the VX8 hardware. This information is extracted with a call to parse_sdf() and the hpvx8 system structure is then created.
4. Finally, the target DSP sites are set up and initialized and group messaging FIFOs are set up between the host and groups via shared memory.

Note This function should be called before any calls are made to hpvx8_group_read() or hpvx8_group_write().

Since this function handshakes with DSP function **`HPVX8_host_msg_setup`** (page 30), the DSP program *must* call the msg_setup function to allow a return. The system_open function does not return until msg_setup has performed its task

`hpvx8_system_close(HPVX8_SYS *sys)`

Arguments: **`sys`** – hpvx8 system structure pointer

Returns: Status

Synopsis: Close system, shutdown VX8 hardware.

Note This function should be called last, after all processing is finished.

`hpvx8_group_read(HPVX8_GRP *grp, PVOID data, PUINT32 length)`

Arguments: **`grp`** – pointer to hpvx8 group
`data` – pointer to buffer where data will be placed
`length` – pointer to length of data read (to be returned)

Returns: Status

Synopsis: Read data from group return messaging FIFO.

Each group has a pair of messaging FIFOs: the *command FIFO* is used to write data from the host to the group DSP; the *return FIFO* is used to read data from the group DSP to the host. The return messages have a *message identifier* word which includes a length field defining the number of words in the rest of the message.

This routine first reads the identifier word, then determines the length of the rest of the message, and then reads the rest of the message. The routine must wait for data

to be available. A time-out timer is used to ensure the routine doesn't hang. The time-out interval can be set with the hpvx8_set_time_out() routine.

Note This routine can be called only after hpvx8_system_open() is called.

`hpvx8_group_write(HPVX8_GRP *grp, PVOID data, UINT32 length)`

Arguments: **`grp`** – pointer to hpvx8 group
`data` – pointer to data buffer to be written
`length` – length of data read (to be returned)

Returns: Status

Synopsis: Write data to group command messaging FIFO.

Each group has a pair of messaging FIFOs: the command FIFO is used to write data from the host to the group DSP and the return FIFO is used to read data from the group DSP to the host. In the hpvx8 scheme, the first word of the data is a command which includes a command field and a data length field. This is followed by *length* data words. A total of *length* + 1 words are written.

This routine first checks the command messaging FIFO for room to place *length* words of data into it. Once room is available, the 'data' is written to the FIFO. Note that time-outs are used to make sure the routine doesn't hang. The time-out interval can be set with the hpvx8_set_time_out() routine.

Note This routine can be called only after hpvx8_system_open() is called.

`hpvx8_error_query(RESULT error, STRING error_msg)`

Arguments: **`error`** – error number to be matched with error message
`error_msg` – string pointer to be filled with error message

Returns: Status

Synopsis: This routine returns a descriptive error message corresponding to the error number passed to it. The error message is obtained by querying the ssvx8 library, the hpvx8 library and finally the SICL or VISA library. In the case of an hpvx8 error, the error message will contain some additional details that are set by the function setting the error. These details are set by calling set_error_detail(). Note that the detail is a static variable so error reporting is inherently non-reentrant.

`hpvx8_set_time_out(FLOAT64 time_out)`

Arguments: **`time_out`** – time out value in seconds

Returns: void

Synopsis: This function is used to set the time-out value for VX8 system accesses. The time-out value is global and is used by all hpvx8 functions that access the system.

Host Programming Example

The following host .c file demonstrates the use of the most commonly used hpvx8 library function calls. This example simply opens a system as defined by the file hpradio.grp (and thus hpradio.sdf and hpradio.sdf), writes a command and reads the reply to that command for each group defined in the system, and then closes the system.

```
/*****************************************************************************/
/*                                                                           */
/*  Example program opens a system, writes the start command to each group,  */
/*  reads each group's reply, and then closes the system.                    */
/*                                                                           */
/*  WindowsNT/95 compile directions                                          */
/*  ———————————————                                        */
/*  For WindowsNT/95, be sure to define _WINDOWS and _VISA (via compiler).   */
/*  The necessary include files can be found in the following directories:   */
/*                                                                           */
/*      C:\vxipnp\winnt\include                                              */
/*      C:\vxipnp\winnt\ssvx8\host\include                                   */
/*      C:\vxipnp\winnt\hpvx8\hpvx8\host\include                             */
/*                                                                           */
/*  This program needs to be linked with hpvx8.lib, ssvx8.lib, sdf.lib,      */
/*  and visa32.lib, all of which can be found in:                            */
/*                                                                           */
/*      C:\vxipnp\winnt\lib\msc                                              */
/*                                                                           */
/*  Note: for Windows95, winnt becomes win95 in the above directory paths.   */
/*                                                                           */
/*  HP-UX compile directions                                                 */
/*  ————————————                                                */
/*  For HP-UX, be sure to define _HPUX and _SICL (or _VISA).  The necessary  */
/*  include files can be found in the following directories:                  */
/*                                                                           */
/*      /opt/vxipnp/hpux/include                                             */
/*      /opt/vxipnp/hpux/ssvx8/host/include                                  */
/*      /opt/vxipnp/hpux/hpvx8/hpvx8/host/include                            */
/*                                                                           */
/*  This program needs to be linked with libhpvx8.a, libssvx8.a, and         */
/*  libsdf.a, which can be found in:                                         */
/*                                                                           */
/*      /opt/vxipnp/hpux/lib                                                 */
/*                                                                           */
/*  Be sure to use the -lsicl (-lvisa) flag to link in the appropriate       */
/*  library.                                                                 */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "hpvx8.h"

void main(void)
{
    RESULT status;                          /* function return status */
    HPVX8_SYS sys;                          /* hpvx8 system structure */
    char error_msg[HPVX8_ERROR_MSG_SIZE];   /* error message buffer */
    HPVX8_GRP *gp;                        /* group pointer */
    UINT32 msg_buf[2];                      /* message buffer */
    UINT32 msg_len;                         /* message length */
```

```
    /* open system with no timeout */
    status = hpvx8_system_open("hpradio.grp", &sys, HPVX8_INFINITE_TIME);
    if (status != SUCCESS)
    {
        hpvx8_error_query(status, error_msg);
        printf("Error during open: %s\n", error_msg);
        exit(status);
    }

    /* send start command to each group defined in system */
    for (gp = sys.group; gp != NULL; gp = gp->next)
    {
        /* send command */
        msg_buf[0] = HPVX8_CMD_Start;
        status = hpvx8_group_write(gp, msg_buf, 1);
        if (status != SUCCESS) break;

        /* read reply from group */
        status = hpvx8_group_read(gp, msg_buf, &msg_len);
        if (status != SUCCESS) break;

        /* is reply what we're expecting? */
        if ((msg_buf[0] != HPVX8_RET_CmdReply) || (msg_buf[1] != SUCCESS))
        {
            printf("Error, group '%s' sent bad reply\n", gp->name);
            break;
        }
    }

    /* report error before closing */
    if (status != SUCCESS)
    {
        hpvx8_error_query(status, error_msg);
        printf("Error while communicating with group '%s': %s\n",
            gp->name, error_msg);
    }

    /* close system */
    hpvx8_system_close(&sys);

    exit(status);
}
```

Target Functions

The following functions make up the hpvx8 target library (see `hpvx8msg.c`).

`hpvx8_host_msg_setup(HPVX8_SITE *site)`

Arguments: **`site`** – pointer to site information, to be copied from host

Returns: Pointer to the host↔site message FIFOs

Synopsis: This routine synchronizes with the host via handshakes to ensure that:

1. the DSP target code gets started at the right time
2. the site setup information is transferred and read at the right time

The pointers to the host↔site messaging FIFOs are derived from the site setup information and returned.

`hpvx8_master_msg_setup(UINT32 site_id)`

Arguments: **`site_id`** – site id of the slave

Returns: Pointer to the master↔slave messaging FIFO

Synopsis: This routine is used to set up messaging between the master (DSP) of a group and one of its slave nodes. Program synchronization between the nodes also occurs.

This routine should be called only by the master node. The slaves should call hpvx8_slave_msg_setup() to synchronize with their master. The master must call this routine once for each of its slaves.

The messaging FIFOs are of the same type as the host↔target messaging FIFOs.

`hpvx8_slave_msg_setup()`

Arguments: void

Returns: Pointer to the master↔slave messaging FIFO

Synopsis: This routine is used to set up messaging between the calling slave (DSP) and its master. Program synchronization between the two also occurs.

This routine should be called only by a slave. A master should call hpvx8_master_msg_setup() to synchronize with its slaves.

The messaging FIFOs are the same type as the host↔target messaging FIFOs.

`hpvx8_msg_cmd_level(HPVX8_MSG *msg)`

Arguments: **`msg`** – address of messaging structure

Returns: Number of slots filled with data in the command FIFO

`hpvx8_msg_ret_level(HPVX8_MSG *msg)`

Arguments: **`msg`** – address of messaging structure

Returns: Number of slots filled with data in the return FIFO

hpvx8_msg_cmd_read(HPVX8_MSG *msg, UINT32 cmd[], UINT32 length)

Arguments: **msg** – address of messaging structure
cmd – pointer to return data, read from FIFO
length – amount of data to be read from FIFO

Returns: void

Synopsis: Reads data from command messaging FIFO.

You must call hpvx8_msg_cmd_level() prior to this command to determine the number of words to read. That is, the data must exist in the FIFO before this routine reads it.

hpvx8_msg_cmd_write(HPVX8_MSG *msg, UINT32 cmd[], UINT32 length)

Arguments: **msg** – address of messaging structure
cmd – pointer to data to be written to FIFO
length – amount of data to be written to FIFO

Returns: void

Synopsis: Writes data to command messaging FIFO.

You must call hpvx8_msg_cmd_level() prior to this command to make sure that there is enough room in the command FIFO to write the data.

hpvx8_msg_ret_read(HPVX8_MSG *msg, UINT32 ret[], UINT32 length)

Arguments: **msg** – address of messaging structure
ret – pointer to return data, read from FIFO
length – amount of data to be read from FIFO

Returns: void

Synopsis: Reads data from return messaging FIFO.

You must call hpvx8_msg_ret_level() prior to this command to determine the number of words to read. That is, the data must exist in the FIFO before this routine reads it.

hpvx8_msg_ret_write(HPVX8_MSG *msg, UINT32 ret[], UINT32 length)

Arguments: **msg** – address of messaging structure
ret – pointer to data to be written to FIFO
length – amount of data to be written to FIFO

Returns: void

Synopsis: Writes data to return messaging FIFO.

You must call hpvx8_msg_ret_level() prior to this command to make sure that there is enough room in the return FIFO to write the data.

Target Programming Example

The radio target code provides an example of using the hpvx8 target functions. Refer to source files in \hpvx8\radio\dsp for details.

DDR Interface Programming

This section describes the sub-directories and files contained in the \vxipnp\winXX\hpvx8\ddrifc directory on WindowNT/95 systems and the /opt/vxipnp/hpux/hpvx8/ddrifc directory on HP-UX systems.

```
dsp/                    # DSP (C4x) target code
     include/           # C4x include directory
         ddr.h          # C4x<->DDR TIM interface header file
     src/               # C4x C source directory
         ddr.c          # C4x<->DDR TIM interface C source file
```

The purpose of each file will be described in sections that follow.

DDR TIM hardware overview

A digital drop receiver usually performs three functions:

1. Channelization and down-conversion of broadband digital data
2. Signal detection or demodulation to extract information
3. Converting information to analog audio output

"Channelization" means to extract a narrowband signal at a specific center frequency from a wideband digital data stream.

The DDR TIM provides functions #1 and #3 with four channels of DDCs and four channels of DACs. The DDR TIM is connected to a C4x DSP processor via a communication port. The C4x DSP processor provides the function #2 portion of the digital drop receiver.

Data Flow

The source of the broadband digital data is a VXIbus digitizer (ADC) such as the HP E1430A (10 Msamples/sec) or HP E1437A (20 Msamples/sec). The ADC sends real (unmixed), 16-bit data to the VX8 via the HP local bus. The HP local bus receiver on the VX8 moves the data into a 1024-word FIFO. This data is then moved to any (or all) of the six TIM sites and the two onboard C40s via a dedicated DMA controller.

The ADC sends the data as two 16-bit samples packed into a 32-bit word. This word is sent to the DDR TIM module in one of the six TIM sites. The two samples are extracted, optionally interleaved with 0-samples[4], and then loaded into each of the four DDC chips. The interleaving and subsequent DDC filter operation is effectively an interpolate-by-2 operation. Interpolation is used with the E1430A to provide 20 Msamples/sec data to the DDC chips. This allows the same decimation steps to be used with both the E1430A and the E1437A since the effective sample rate seen at the DDC input will be 20 Msamples/sec for either ADC.

[4] Depends on the ADC installed. Not necessary with the E1437A.

The DDC chip is the Harris Semiconductor HSP50016 digital downconverter. Each of the four DDCs accepts data from the same ADC data stream, mixes it with a unique programmable complex center frequency, and decimates the resulting complex signal to provide a 24-bit complex, channelized, and downconverted output signal. The decimation rate must be the same for each DDC on the module. A gate array on the DDR TIM module collects the output signals from each of the four DDCs, multiplexes them together, and then sends the collective data stream out a communication port to a C4x DSP.

The comm port interface (based on the C4x comm port) provides bi-directional data flow. Data from the DDCs is sent to the C4x while DDC control words are sent from the C4x to the DDR TIM module and then routed to the DDC chips on the module. The DDC control words allow the C4x DSP to set DDC parameters such as center frequency, decimation rate (or IF bandwidth) and gain, among others.

The channelized data from the DDCs are multiplexed and sent to the C4x in the following order:

```
channel 0 real
channel 0 imaginary
channel 1 real
channel 1 imaginary
channel 2 real
channel 2 imaginary
channel 3 real
channel 3 imaginary
```

This process repeats for each sample point. The data is sent in a 32-bit word (as per the C4x comm port specification) and contains a 24-bit real or imaginary sample point along with some identification information. See the **DDR Communication Formats** section (following) for more details. The DDR TIM module can be programmed to output data from either all 4 channels or just the first two channels, channels 0 and 1. The C4x should be ready to accept data from the DDR TIM (via its comm port) without delay since data can be missed if not read in a timely manner. As long as the C4x reads its input comm port FIFO such that the FIFO never becomes full, no data will be lost. Setting up an internal C4x DMA process to transfer data from the comm port to memory is an efficient way to manage this data flow. The 'radio' target example shows one way of doing this.

More information about the Harris Semiconductor HSP50016 DDC chip can be found at Harris' website, www.semi.harris.com under the 'Digital Signal Processing' link.

Comm Port Operation

The comm port interface is used to pass data and control words between the C4x DSP and the DDR TIMs. The operations allowed are as follows:

- send data from the DDCs to C4x DSP
- send data from the C4x DSP to the output audio DACs
- send control information to the DACs
- program the DAC decimation counter
- send control information to the DDCs
- configure the operation of the DDR TIM module

The data and control word formats are detailed in the **DDR Communication Formats** section (following).

After the C4x DSP receives the channelized data and detects or demodulates it, it can optionally send the audio information contained in the signal back to the DDR TIM module's onboard audio DACs. Each of the four DACs is sent a 16-bit real audio data stream in the format described in the **DDR Communication Formats** section. The data must always be sent in channel pairs (i.e. channels 0 and 1, and then channels 2 and 3). Even if only one audio channel is being utilized, its paired channel must have data sent to it.

The DAC has a decimation counter associated with it to provide control over the output sample rate. The counter divides the ADC sample rate by the number it is programmed to. The ADC clock and the DAC clock must be synchronized to ensure that data flows at the appropriate rate through the system. If these clocks were not derived from the same source, data starvation or overflow could occur. The DAC decimation counter should be programmed with the rate equal to the decimation rate of the DDCs, scaled up (or down) for any interpolation (or decimation) that the C4x DSP program performs in its processing. The audio DACs accept any DAC sample rate within the octave 16.276 kHz to 32.552 kHz.

For the demodulated audio signal data rate to fall within this octave, the C4x program must interpolate or decimate the DDC's output sample rate by the appropriate factor of two. For example, if the system is using an E1437A with a sample rate of 20.48 MHz and the DDCs are programmed to decimate by 2048, the DDCs send a stream of 10 kHz (10 ksa/sec) data to the C4x for detection. Somewhere in the processing flow the C4x program must interpolate this data stream by 2 to get its data rate to fall within the DAC's allowed octave.

The DACs' analog audio output signals are buffered and connected to the VX8's front panel via the application specific connector. The mapping of each TIM site's audio outputs and the front panel connector pins is provided in the table on page 7.

DDR Communication Formats

The DDR data and control word formats are provided below. Each data word is 32-bits wide, as specified by the C4x comm port.

- DDC data word (from DDR to DSP)

```
bits 31- 8: 24-bit real or imaginary data, 2's-complement
bits  7- 3: copy of tag field in control register, allows DDR id
bits  2- 1: indicates data came from DDC channel number (0-3)
bit       0: indicates data is real (0) or imaginary (1)
```

Bits 7-0 are forced to 0 unless the DDCINFO flag in the control register is set to 1.

- DAC data word format (from DSP to DDR)

```
bits 31-16: 16-bit 2's-complement audio data
bits 15- 8: set these bits to 0
bits  7- 4: command id (0x1)
bits  3- 2: set these bits to 0
bits  1- 0: the DAC ch. to which audio data should be routed (0-3)
```

- DAC control word format (from DSP to DDR)

```
bits 31-24: set these bits to 0
bits 23-16: stereo DAC control word, where:
    bit     23: if 0 select channel 0|2, else select channel 1|3
    bit     22: if 1 mute output
    bits 21-16: attenuation in dB (0-63)
bits 15- 8: set these bits to 0
bits  7- 4: command id (0x2)
bits  3- 1: set these bits to 0
bits      0: if 0 select DAC channels 0 and 1, else channels 2 and 3
```

The combination of bits 0 and 23 are used to select the channel to which the mute and attenuation parameters are sent:

bit 0	bit 23	DAC channel
0	0	0
0	1	1
1	0	2
1	1	3

- DAC decimation counter value (from DSP to DDR)

```
bits 31-25: set these bits to 0
bits 24-16: 9-bit DAC decimation count-1 (clock divider)
bits 15- 8: set these bits to 0
bits  7- 0: command id (0x22)
```

See the `ddr.h` and `ddr.c` files for more information.

- DDC control start command (from DSP to DDR)

```
bits 31- 8: set these bits to 0
bits  7- 0: command id (0x44)
```

This control command is required to start off the DDC control word sequence. To write to a particular DDC, write this word first, then write five consecutive DDC control bytes to make a 40-bit DDC control word value.

- DDC control word format (from DSP to DDR)

```
bits 31-24: set these bits to 0
bits 23-16: control byte, one of 5 in the 40-bit DDC control word
bits 15- 8: set these bits to 0
bits  7- 4: command id (0x4)
bits  3- 2: set these bits to 0
bits  1- 0: select DDC channel (0-3)
```

For more information on the 40-bit DDC control words, see the Harris Semiconductor HSP50016 DDC data sheet.

- DDR control register (from DSP to DDR)

```
bits 31-16: DDR control word, where:
   bits 31-29: set these bits to 0
   bits 28-24: tag field value gets inserted in bits 7-3 of each
               DDC data word sent to C4x if DDCINFO is set to 1
   bit     23: set this bit to 0
   bit     22: DATAEN enables the data flow from DDC to comm port
               (and then to DSP) when set to 1
   bit     21: DDCINFO causes the DDC data information to be placed
               in the lower bits of each data word when set to 1
   bit     20: SYNCEN enables the 'DAC ready for data' interrupt
               signal to DSP when set to 1
   bit     19: CH2 limits the DDC output to two channels when set
               to 1, four channels when cleared to 0
   bit     18: INTERP activates the interpolator (L=2) when set to 1
               (to be used with E1430A ADC)
   bit     17: DACRST* resets the DAC circuitry when cleared to 0
   bit     16: SRST* resets the entire DDR TIM when cleared to 0
bits 15- 8: set these bits to 0
bits  7- 0: command id (0x80)
```

Each bit in the control register (bits 31-16) is cleared to 0 on power-up, iclear vxi, or hard reset (VX8 reset).

DDR Data Structures

To simplify the task of programming the DDR TIM module, a DDR library is provided (see `ddr.c`). The state of a DDR TIM module is contained in a structure of type DDR_STATE (see `ddr.h`). This structure has the following DDR TIM parameters:

- the C4x communication port the DDR TIM is connected to (`comm_port`)
- the DDC decimation value (`ddc_decimation`)
- the DAC decimation value (`dac_decimation`)
- the state of the DDR TIM control register SRST* bit (`reset`)
  0 ⇒reset, 1 ⇒not reset
- the state of the DDR TIM control register DACRST* bit (`dac_reset`)
  0 ⇒reset, 1 ⇒not reset
- the state of the DDR TIM control register INTERP bit (`interpolate`)
  0 ⇒no interpolation (for E1437A), 1 ⇒interpolate (for E1430A)
- the state of the DDR TIM control register CH2 bit (`number_channels`)
  0 ⇒4 channels, 1 ⇒2 channels
- the state of the DDR TIM control register SYNCEN bit (`sync_enable`)
  0 ⇒disable, 1 ⇒enable
- the state of the DDR TIM control register DDCINFO bit (`ddc_info`)
  (turns on tag, channel, & real/imag information bits in DDC words)
  0 ⇒'00000000', 1 ⇒'tttttccr', where tttt ⇒tag, cc ⇒channel, r ⇒real(0)/imag(1)
- the state of the DDR TIM control register DATAEN bit (`data_enable`)
  0 ⇒disable, 1 ⇒enable
- the DDR TIM control register tag field (`data_tag`). A 5-bit, user-defined tag placed in the lower byte of the DDC data (bits 7–3). Active only when the `ddc_info` bit is set.
- a set of four channel-specific parameters (`DDR_CHANNEL`):
  – the DDC center frequency (phase_increment)
  – the DDC spectrum type, up convert or normal (spectrum_type)
  – the DDC spectrum reversal flag (spectrum_reverse)
  – the DAC mute state (mute)
  – the DAC attenuation value (attenuation)

The DDR library functions are used to control and communicate with the DDR TIM module and use the DDR_STATE to determine which parameters to set. Normally the desired DDR_STATE parameter is changed and then the appropriate function is called to make this change to the DDR TIM hardware. A list of DDR library functions follows:

DDR Functions

Typically, the only functions required to setup and control DDRs are `ddr_new`, `ddr_init`, `ddr_delete`, and `ddr_set_all`. Functions used while running are `ddc_center`, `dac_control`, and `dac_set`. See example code for typical use.

`ddr_new(long comm_port)`

Arguments: **`comm_port`** – C40 comm port to which the relevant DDR is attached

Returns: Pointer to the DDR state structure created.

Synopsis: This function creates a new DDR_STATE structure and initializes it with known reset values of the DDR TIM module. This function is called first.

Note `ddr_init()` should be called after `ddr_new` to initialize the DDR hardware.

`ddr_init(DDR_STATE *ddr)`

Arguments: **`ddr`** – Pointer to DDR state structure (used to identify comm port)

Returns: void

Synopsis: This function initializes the DDR TIM hardware to be ready for a call to ddr_set_all(). It clears out the DAC buffers and initializes each DDC on the DDR with a write to control word 5. This routine should be called second, after ddr_new().

Notes `ddr_init()` should be called immediately after `ddr_new` to initialize the DDR hardware.

Upon exit `ddr->`reset = RESET and `ddr->dac_reset = DAC_UNRESET`.

`ddr_set_all(DDR_STATE *ddr_list[], long length)`

Arguments: **`ddr_list[]`** – array of pointers to DDR state structure of DDRs to be programmed
`length` – number of DDRs on `ddr_list[]`

Returns: void

Synopsis: Sets up and programs all parameters on all DDR TIMs listed in its argument. This routine first initializes the DACs of each DDR by calling `dac_set()`. The DDRs are then un-reset in reverse order in a tight loop so that the DAC clocks for each DDR are fairly close to each other. It is assumed that the first DDR in the passed-in list will be the one that generates the DAC service interrupt; the order of module un-resetting guarantees that the other DDRs will be ready to accept new DAC data when the interrupt is generated. The rest of the DDR (and DDC) parameters are set by calling `ddr_set()` for each DDR in the list. This function should be call third, after `ddr_new()` and `ddr_init()` has been called for each DDR on the list.

ddr_set(DDR_STATE *ddr)

Arguments: **ddr** – pointer to DDR state structure (used to identify comm port)

Returns: void

Synopsis: Sets all DDR and DDC control words on a DDR TIM module, except for the parameters set in dac_set(), according to the DDR_STATE for the module. Again, all of the fields of the DDR_STATE structure should be changed to their desired values before calling this routine. The state of the DDR TIM hardware will match the DDR_STATE after this routine completes. It is usually called after `dac_set()`. See `ddr_set_all()`.

Note It is easier to call `ddr_set_all()` to set up all of the DDRs in the correct order.

ddr_control(DDR_STATE *ddr)

Arguments: **ddr** – pointer to DDR state structure (used to identify comm port)

Returns: void

Synopsis: Updates the DDR control word from the parameters in the DDR_STATE structure. This function is called by `ddr_set_all()`.

ddc_ifbw_and_spectrum(DDR_STATE *ddr)

Arguments: **ddr** – pointer to DDR state structure (used to identify comm port)

Returns: void

Synopsis: Changes the decimation rate (IF bandwidth) and spectrum type (up convert, spectral reverse) for each of the 4 (or 2) DDCs on the specified DDR. All DDCs on the DDR will be updated to reflect the IF bandwidth and spectrum parameters of the DDR_STATE structure.

Notes It is easier to call `ddr_set_all()` since all DDRs must use the same IF BW.

The desired decimation and spectrum values should be set in the DDR state structure *before* calling this function.

ddc_center(DDR_STATE *ddr, long channel_number)

Arguments: **ddr** – pointer to DDR state structure (used to identify comm port)
channel_number – DDC channel number (range 0..3)

Returns: void

Synopsis: Changes the phase increment (LO center frequency) of the specified DDC channel on a given DDR.

$$\text{ddr\_phase\_increment} = \frac{F_c}{F_s} \times 2^{33}$$

where F_c is LO frequency and is F_s incoming sampling frequency.

This can be used "on the fly" while the DDC is operating.

dac_set(DDR_STATE *ddr)

Arguments: **ddr** – pointer to DDR state structure (used to identify comm port)

Returns: void

Synopsis: Initializes the DDR DAC by writing zero samples to the DAC buffers and by writing the DAC decimation counter (clock divider) value. Usually called before `ddr_set()`. See `ddr_set_all()`.

Note It is easier to call `ddr_set_all()` since it calls associated routines in the correct order.

dac_control(DDR_STATE *ddr, long channel_number)

Arguments: **ddr** – pointer to DDR state structure (used to identify comm port)
channel_number – DAC channel number (range 0..3)

Returns: void

Synopsis: Sets the DAC control bits including mute and attenuation values for the specified channel.

This can be used "on the fly" while the DAC is operating.

dac_decimate(DDR_STATE *ddr)

Arguments: **ddr** – pointer to DDR state structure (used to identify comm port)

Returns: void

Synopsis: Sets DAC decimation (clock divider) value. The decimation value range is 0..511.

$$\text{decimation} = \frac{dec}{4} - 1$$

where ***dec*** is the desired decimation rate from either 20 MHz (the sample rate of the E1437A) or 20.48 MHz (the interpolated sample rate of the E1430A).

This function is called by `dac_set()` which is, in turn, called by `ddr_set_all()`.

dac_out(DDR_STATE *ddr, long channel_number, long data)

Arguments: **ddr** – pointer to DDR state structure (used to identify comm port)
channel_number – DAC channel number (range 0..3)
data – 16-bit, signed integer (range -32768..32767)

Returns: void

Synopsis: Outputs audio data sample to specified DAC channel. Normally this routine is called from a DAC interrupt service routine.

ddr_delete(DDR_STATE *ddr)

Arguments: **ddr** – pointer to DDR state structure

Returns: void

Synopsis: Returns a structure of type DDR_STATE to the memory heap.

DDR Programming Example

The following C4x .c file demonstrates the use of the DDR library function calls. This example simply creates a DDR_STATE structure for one DDR connected to comm port 5. It then initializes and sets the DDR TIM hardware. After some processing the center frequency of channel 0 is changed.

```
/*************************************************************************/
/*                                                                       */
/*  Example program demonstrates the use of the DDR library functions.   */
/*                                                                       */
/*  compile directions                                                   */
/*  _________                                                  */
/*  Compile ddr.c and link with this program.  Make sure the following   */
/*  directory is in the include path:                                    */
/*                                                                       */
/*      C:\vxipnp\winnt\ddrifc\dsp\include                               */
/*                                                                       */
/*************************************************************************/

#include "ddr.h"

#define POW2_33 8589934592.0    /* 2^33 */

void main(void)
{
    DDR_STATE *ddr;             /* DDR state structure */

    int i;                      /* loop counter */

    /* create new DDR state structure for DDR connected to comm port 5 */
    ddr = ddr_new(5);

    /* initialize DDR hardware for programming */
    ddr_init(ddr);

    /* set state of DDR, note reset and dac_reset were set by ddr_init()
    ____________________________________ */
    /* DDCs decimate by 2048 -> 10 kSamples/sec, IF bandwidth = 5.5 kHz */
    ddr->ddc_decimation = 511;

    /* DACs decimate by 1024 -> 20 kSamples/sec */
    ddr->dac_decimation = 255;

    /* E1437A is being used, 20.48 MSamples/sec */
    ddr->interpolate = NO_INTERPOLATE;

    /* all four DDC channels will be used */
    ddr->number_channels = CHANNELS_4;

    /* don't enable DAC interrupt until ready to output audio data */
    ddr->sync_enable = SYNC_DISABLE;

    /* enable tag, channel, and real/imaginary information in the data */
    ddr->ddc_info = DDC_INFO;

    /* enable data to flow from DDCs immediately */
    ddr->data_enable = DATA_ENABLE;
```

```
    /* set DDR identification tag */
    ddr->data_tag = 1;

    /* set parameters for each channel */
    for (i = 0; i < NUM_DDR_CHANNELS; i++)
    {
        /* set phase increment = (2^33)*Fc/Fs to 'i' MHz */
        ddr->channel[i].phase_increment = POW2_33 * i / 20.48;

        /* use normal spectrum */
        ddr->channel[i].spectrum_type = NO_UP_CONVERT;

        /* use no spectral reversal */
        ddr->channel[i].spectrum_reverse = NO_REVERSE;

        /* no audio mute */
        ddr->channel[i].mute = NO_MUTE;

        /* full volume */
        ddr->channel[i].attenuation = 0;
    }

    /* set all DDR hardware parameters and start up hardware */
    ddr_set_all(&ddr, 1);

    /*** do some processing here ***/

    /* set channel 0 frequency to 950 kHz */
    ddr->channel[0].phase_increment = POW2_33 * 0.950 / 20.48;
    ddc_center(ddr, 0);

    /*** do some more processing here ***/

    /* return ddr memory to heap */
    ddr_delete(ddr);
}
```

HPVX8 Host Library

For WindowsNT/95 systems, you must have Microsoft Visual C++ 5.0 to use the winnt.mak and winnt.dsp files (in the hpvx8\host\build directory) directly.

Note

Although the makefile is called 'winnt', it works for both WindowsNT and Windows95. In fact, the hpvx8 library works on either platform regardless of the platform it was built on.

Once Microsoft Visual Studio comes up, select the 'Build->Rebuild All' menu pick to rebuild the library. If another host compiler is being used, compile all of the hpvx8\host\src files with the following preprocessor definitions:

```
WIN32, _WINDOWS, and _VISA.
```

Include paths require pointers to the ssvx8 and VISA include directories. The resulting object files can be linked directly with the host application files or built as a library, either shared or static. Note that the hpvx8 library requires the ssvx8 library as well as the VISA library to function.

For HP-UX, simply use the following command to make both the shared and archive (static) libraries:

```
make -f hpux.mak
```

The HP-UX version of the hpvx8 library requires the ssvx8 shared library as well as the SICL or VISA shared libraries (depending on the HP-UX version) to run with an application program.

HPVX8 Target Program

To use the target side of the hpvx8 library, compile hpvx8msg.c and link with the C4x application program. See the radio build instructions on page 61 for more information.

HP Radio Target Program

The VX8 Opt. 140 AM/FM/Sideband Multi-channel Radio (HP Radio), is a system demonstration tool for the HP VX8 VXI DSP board (VX8) and the companion VX8 Opt. 040 DDR TIM module.

HP Radio is a programming example demonstrating the use of the VX8 and DDR TIM to create a multi-channel, narrowband receiver system. The system is comprised of two main parts: the host (controller) and the target (DSP).

The target portion of HP Radio ("Radio") is an example application, with source code, illustrating concepts necessary for the successful development of a VX8 DDR system. This document describes the Radio DSP example software, providing an overview and discussing the relationship between the software and constraints presented by the hardware.

Reference Documentation

This discussion assumes that you have access to and are familiar with:

- the VX8 documentation, in particular the Technical Reference and the Programming Guide
- the TI TMS320C4x processor family and its documentation, in particular the C4x User's Guide.
- the TI C4x code development tools and their associated documentation, in particular the Optimizing C Compiler User's Guide, the Assembly Language Tools User's Guide, and the Parallel Runtime Support Library User's Guide.

Data Flow

Overview

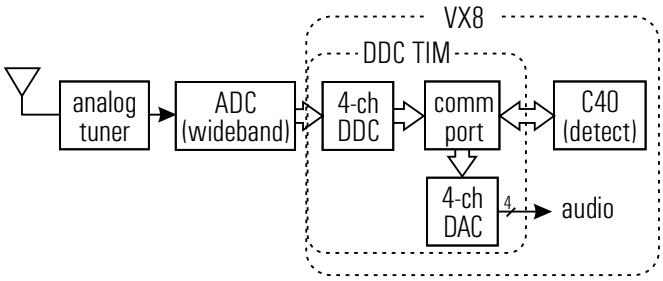
Information flows from an antenna through the system front end to the VX8 board where the narrowband receiver functionality exists. The process is described in the following discussion by tracking the data flow through the various steps.

- The VXI system overview.
- The VX8 carrier board moves ADC data from the HP local bus to the DDR TIM
- The DDR boards perform digital downconversion and decimation.
- C4x flow and digital processing
- Data flows back through the comm port to the DACs on the DDR TIM which converts the digital audio to analog signals.

VXI System

The data flow through the system is illustrated by the figures below. At the very highest level, a system ADC module to the left of the VX8 acquires data, possibly from an RF downconverter, and sends it to the VX8 over *local bus*. This unidirectional high-speed bus between adjacent HP VXI modules is also called *HP local bus*. The VX8, with its C4x processors and DDR TIM modules, then tunes to specific frequency bands, demodulates the signal within each band, and outputs the demodulated audio.

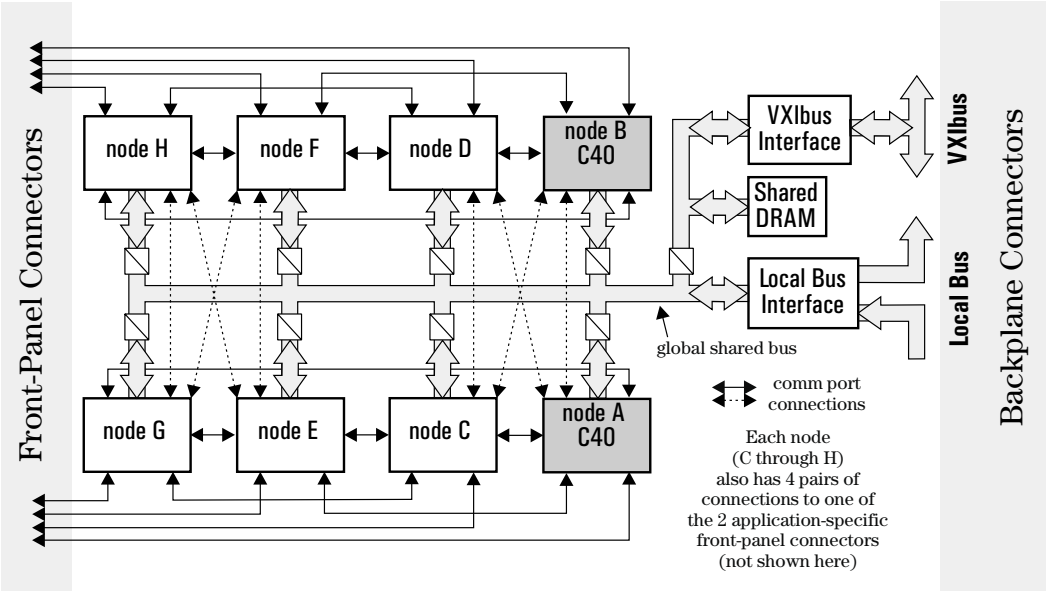
The figure below illustrates how data moves back and forth between the DSP and the DDR modules; one comm port is used to move data from the DDC to the DSP and then from the DSP to the DAC.



The following discussions cover these steps in more detail.

VX8

On the VX8 carrier board, the HP local bus interface receives the system ADC data, and then HP local bus DMA sends it to all or a particular subset of the eight nodes on the VX8. (See Setting DMA Target on page 50.) These nodesconsist of two embedded C4x sites A and B, and six TIM sites C through H. In an HP Radio system, DDR TIM modules or additional C4x processor modules are plugged into some or all of the TIM sites, and the DDR TIM modules receive the HP local bus DMA data for digital downconversion. C4x nodes can also receive this data for other processing such as computing its spectrum.



The next discussion covers data flow on the DDR TIM board in more detail.

DDR TIM

On the DDR TIM board, logic receives the wideband ADC data at the global bus connector, unpacks it (each 32-bit word contains two 16-bit samples), hardware-interpolates it by two if selected, and routes it to the four digital downconverters (DDCs). These are Harris Semiconductor HSP50016 digital downconverters.

Downconversion The DDCs perform digital downconversion in two steps.

1. First, the incoming real data stream is multiplied by a digital quadrature local oscillator, generating a complex representation with the desired tuned frequency shifted to DC.
2. Then, low-pass filtering and decimation are performed on the real and imaginary parts of this complex data stream, reducing its bandwidth and sample rate to the required extent to select the desired channel.

Decimation The DDCs have flexible decimation rates. Decimation is by a factor of 4×R, where R can range from 16 to 32768, inclusive, in integer steps. This gives a minimum decimation of 4×16 or 64, and a maximum decimation of 4×32768 or 131072. Given an E1437A ADC running at 20.48 MHz, or an E1430A ADC running at 10.24 MHz with the hardware interpolator selected, the input rate to the DDCs is 20.48 MHz, and a DDC's (complex) output rate can range from 20.48 MHz / 64 or 320 kHz, down to 20.48 MHz / 131072 or 156.25 Hz.

The 3 dB bandwidth at the output of a DDC is 0.55 times its (complex) output sample rate. Thus bandwidths from 176 kHz down to 85.9 Hz are possible. While these bandwidths are possible given the DDC hardware, the Radio software is restricted to a useful subset of these bandwidths for two reasons.

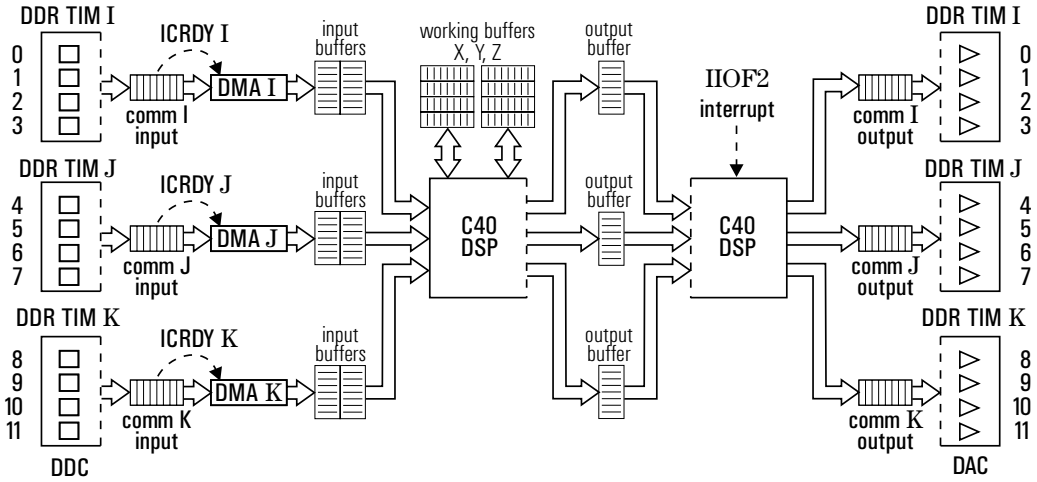
1. A power-of-two relationship is required between the DDC rate and the DAC rate, precluding some bandwidth choices.
2. Some combinations of bandwidth, number of channels being processed, and demodulation algorithm complexity preclude real-time operation due to the time between blocks of input data being too short to accomplish the calculations.

DDC data flowing from the DDR TIM to the C4x consists of formatted DDC output data. These are 32-bit words consisting of the 24-bit raw DDC output words concatenated with a least-significant-byte identification field. This is shown on the left side of the next figure.

The design of the DDR TIM hardware ensures that all DDCs on a DDR TIM operate at exactly the same bandwidth and thus output sample rate. This makes possible the data interleaving that must occur among the DDC output data. The HPRadio program further requires that all DDR TIM modules in a group operate at the same bandwidth. This is due to the fact that the C4x only has one interrupt input available for output data (see the DAC ISR discussion on page 58).

C4x Data Flow

The figure below illustrates the flow of data from the perspective of the Radio target program running on the C4x DSP. At the left, DDC data from the DDR TIM enters on the comm port and the C4x DMA coprocessors transfer it to double buffers in RAM. This is complex data representing tuned bands in the ADC input (or RF input in the case of a downconverter). Demodulation is done on the data streams, and the resulting real audio data is returned to the DDR TIM via the same comm port used to input the original data. See also, **Process** on page 52.



As many as 3 DDR TIM modules per C4x (12 channels/DSP) are supported by the Radio program as illustrated in the figure above. The DDR TIMs are labeled I, J, and K. The DDC portion of each DDR TIM is shown at the left side of the figure as the source of the input data; the DAC portion appears at the right as the destination for the demodulated data. The following discussion covers this in more detail.

C40 DMA DDC data is brought into the C4x environment from a DDR TIM via a comm port and its associated DMA coprocessor. The C4x DMA coprocessors reduce the C4x processor's overhead in this task to nearly zero by continuously and automatically loading data into an input double-buffer, auto-initializing to the opposite side of the buffer when one side becomes full. To keep the coprocessors from tying up internal C4x busses between input samples, the program uses ICRDY interrupts between the comm ports and their associated C4x DMA coprocessors. This is called *source synchronization mode*.

Real-Time The DMA coprocessor and the C4x communicate as follows:

- A bit in the C4x IIOF flag (IIF) register indicates DMA transfer complete.
- The C4x DMA coprocessor destination address register (memory-mapped to the C4x processor) indicates which side of an input buffer has the most-recent data.

The C4x DSP process always checks the flag and DMA address register to determine when and where to read input data. When the process is operating real-time, the data is read alternately from the two buffers and the process has to wait for a buffer to finish filling[5] (indicated by the flag) before it can start processing the next block. When the process drops out of real-time, the process may skip one or more buffers of data on each cycle[6].

[5] For a group with more than one DDR TIM, waiting for *any* of the group's input buffers to fill indicates real-time operation.[5]

[6] Also, the process does not have to wait to read data, which indicates non-real-time operation. Non-real-time operation is indicated by lighting the User (amber) LED on the VX8 front panel.

C4x DSP From the input buffers, DDC data flows into a chain of C4x DSP operations. A set of floating-point, working buffers called X, Y, and Z are used to hold interim results. Data is processed from one DDR TIM at a time. The X, Y, and Z working buffers are each a set of four, one for each channel in a DDR TIM. See the center section of the figure on page 47. The DSP operations are discussed further under Process on page 52.

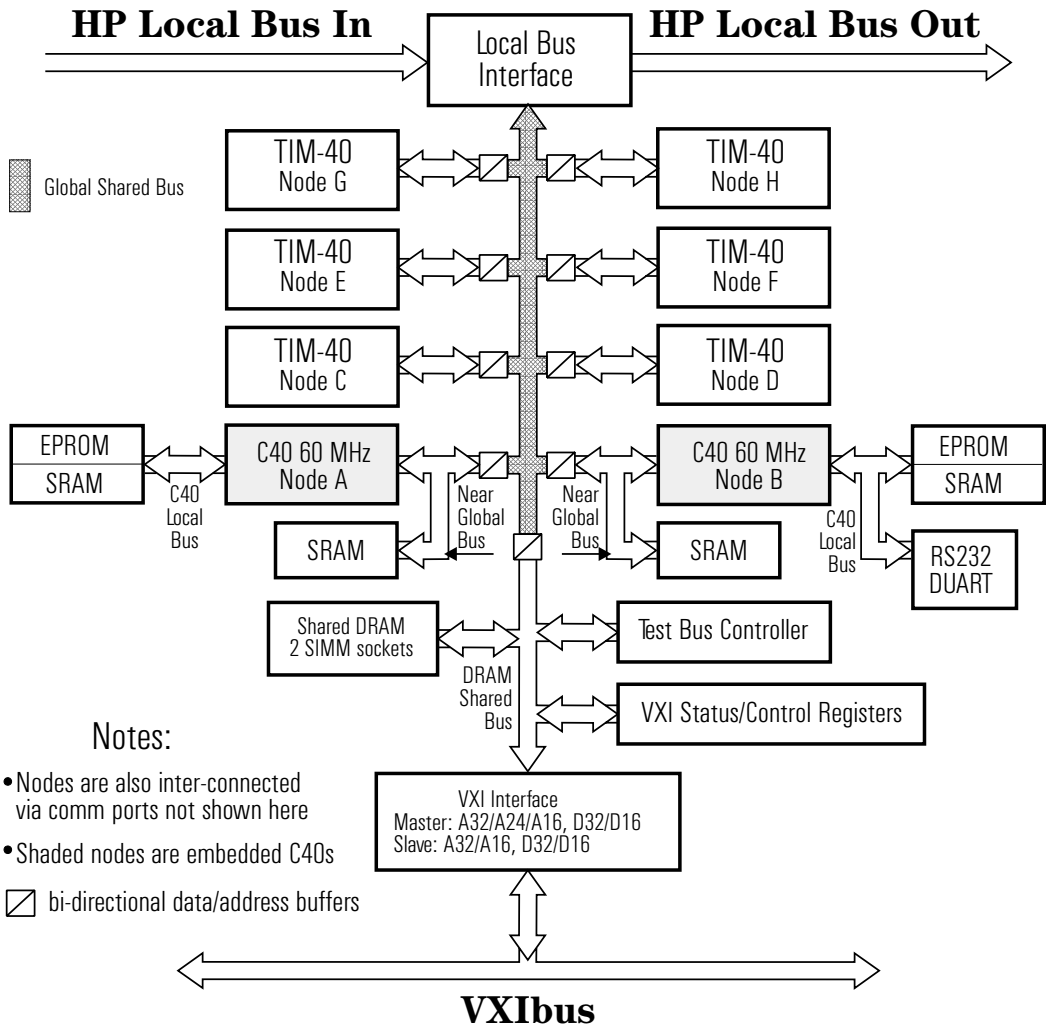
At the end of C4x DSP operations, the demodulated audio output data is deposited into output buffers, one buffer per DDR TIM, with each 32-bit word containing a 16-bit sample and identification information.

Digital Audio Output

From the output buffers, data flows through the comm port back to DDR TIM output DACs (digital-to-analog converters). For more information about the DAC interrupt service routine which performs this task see **DAC ISR** on page 58.

VX8 Shared Bus

This discussion covers the operation/use of the Global Shared Bus which is an important component of the DDR system. See the figure below. Also, more information is available in the *VX8 Carrier Board Technical Reference* from Spectrum Signal Processing under Bus Architecture.



The HP local bus DMA uses the global shared bus to move ADC data from the HP local bus interface to the near global memory space of the embedded and/or TIM nodes. Moving ADC data is the main task for the global shared bus but it is also used to move host commands from DRAM to the DSPs.

The ADC data throughput rate can be quite large. For example, with an E1437A ADC running at a 20.48 MHz sample rate, the average data rate for packed, 32-bit words is 10.24 MHz, equivalent to 40.96 megabytes per second; a significant fraction of the capability of the global shared bus. The DMA locks the global shared bus while it broadcasts data to the nodes. DSPs trying to access the bus during the DMA broadcast (to get host commands from DRAM) must wait until the DMA gives up the bus.

Passing Commands

Host commands to the target program(s) running on the DSPs are read from DRAM to the DSP's near global RAM via the global shared bus. To get access to the global shared bus, we use the gap interrupt, as follows.

- The host defines the block size for the system ADC.
- The ADC sends an end-of-block (EOB) signal that identifies the last block of data in its output data stream. For Radio, the typical block size is 65,536 samples (32,768 double-sample words). This block size should not be confused with the C4x DMA, which operates with its own block size on post-DDCata.
- When the HP local bus interface receives an EOB signal, it halts the flow of incoming data and generates an interrupt.[7] All C4x sites running Radio service this interrupt with what is called the *gap* ISR, so named because it represents a gap in the input data flow. The gap ISR is the *only* opportunity for the C4x processors to use the global shared bus without risk of halting until the bus becomes free.
- At the end of the gap ISR, the node A program clears the interrupt, allowing the input data to resume flowing.

Input data accumulates in the ADC FIFO while the gap ISR runs. The ADC halts when its FIFO becomes full, so it is important that the radio program executes the gap ISR quickly.

For more information, see Gap ISR on page 58.

Setting DMA Target

To allow a C4x node maximum access to its own near-global assets, you should limit the HP local bus broadcast DMA to only those nodes that require the ADC data. A C4x node can always access its own near global memory if no other user is accessing it, even when the global shared bus is busy. However, if another process is accessing a node's near global memory, the node's access is halted until the other access ends.

The broadcast DMA target parameter specifies which nodes receive the input data. See the `VX8_HPSetDMATarget` command in the *VX8 Carrier Board Programming Guide*. The following targets are allowed:

- All nodes
- Any individual node
- Nodes C, D
- Nodes C, D, G, H
- Nodes E, F, G, H
- Nodes C, D, E, F, G, H

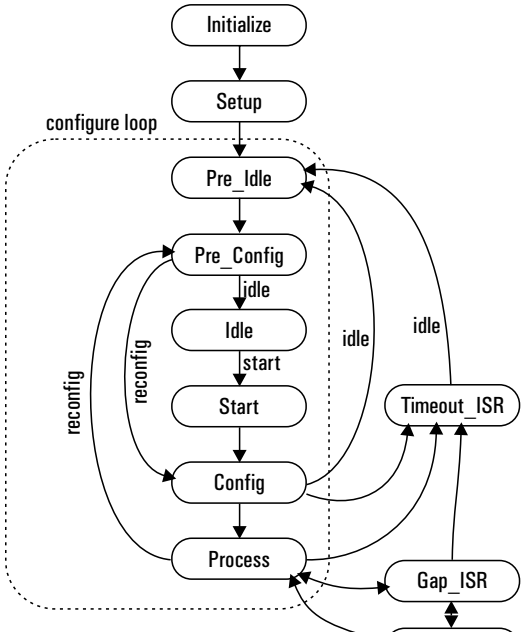
Limiting the broadcast to only the nodes that process input data allows the other nodes to have free access to their near global bus memory. This is significant because performance is maximized by distributing the DSP data and program elements among as many of the C4x data busses as possible: internal, local external, and global external.

For Radio, the ADC data is used by all the DDR TIM nodes. A C4x node needs raw input data only if that site is computing a spectrum from the data. For configuration simplicity, Radio obeys this constraint, using only internal and local external memory. The linker command files could be edited to make use of this memory, precluding the spectrum computation. See the section on C4x memory utilization on page 59.

[7] The HP local bus also halts the incoming data flow and releases the global shared bus when its 1K FIFO is "almost empty". The gap interrupt, however, occurs only when the EOB signal is received.

Control Flow

The following figure is the program state diagram for the Radio C40 target program.



Initialize

Variable initialization is done in the initialize state and node A initializes the memory which is used for C4x-to-C4x interprocessor synchronization.

Note

Global compound variables (arrays and structures) should be initialized in code. The code generation tools for the VX8 do not initialize the .bss section containing global and static local data to zero in the absence of explicit initializers. There is a workaround involving the fill command in the linker command file, but it increases the size of the executable COFF .out files and increases the load time.

Setup

A host handshake is performed, and information is retrieved from the host about host messaging and site configuration. The host must handshake with node A before any other site to guarantee that node A has finished initializing the interprocessor synchronization memory before another node tries to use this memory. Node configuration information is used to tell the program its site offset for interprocessor synchronization.

Next, node A gets a required initial command from the host giving additional information on system configuration, specifically which other C40s in the system are also running Radio and thus will be involved in interprocessor synchronization. Site A then sets up the DDR TIM to C4x DAC interrupt connection(s).

Finally, the interrupt vector table is set up.

Pre-Idle

First in the configure loop is the pre-idle state. Here the VX8 local bus DMA is reset and prepared for restarting, but not yet restarted. This stops the flow of system ADC input data, freeing the global shared bus.

Pre-Config

During the pre-config state, the DDR TIMs are reset. In concert with this, C4x DAC and DMA interrupts are disabled and process state working variables are cleared. Passing through this state in reconfig, system ADC data continues to flow to the global memory interface of the DDR TIMs, but is ignored, since the DDR TIMs are reset. See **Leaving Process** below.

Idle

In the idle state, host commands from DRAM are processed, setting up program variables or the DDR TIM state structure. This same fetch-execute cycle occurs during the process state except that, in the process state, the fetch occurs in the GAP ISR.

Command processing continues until a start command is received.

Start

The start command moves program execution into the start state. The HP local bus DMA is restarted and the time-out timer is started. See Timeout ISR on page 58.

Config

The config state sets up the C4x DMA, reflecting any change in C4x DMA block size. The C4x DAC interrupt is reenabled, and then a signal is sent to the gap ISR to unreset and set up the DDR TIMs the next time the gap ISR occurs. The config state is exited after a signal comes back from the gap ISR indicating that the DDR TIM unreset and setup has occurred.

Process

The core of the Radio program is the process state. The processing described in C4x Data Flow (page 47) occurs here. This state involves a while loop called the process loop at the end of the main( ) function of radio.c The process loop has exactly one exit. One C4x DMA block for each DDR is processed each time through this state.

At the top of the process loop, host commands are processed. The gap ISR fetches host commands from a message buffer in DRAM to the DSP over the global shared bus[8]. The gap ISR then sets a flag indicating that a command is available. Commands are processed by the `commandInterface()` function.

The only loop exit follows immediately and is based on results of the command processing.

Leaving Process

Execution may leave the process state in two ways: reconfig, and idle. Reconfig involves temporarily leaving the process state, going through the pre-config and config states, and then reentering the process state. During reconfig, the system ADC continues to run and gap interrupts continue to occur. The primary purpose of reconfig is to handle changes to the C40 DMA block size due to commands such as setting the IF BW.

The other way to leave the process state is to go idle. Idle is an exit back to the idle state, where control remains until the start command is received. Idle also represents how the program first starts up after the initialize and setup states.

[8] See following discussions for more detail: Process on page 52 and Gap ISR on page 58.[8888]

Next, input data is processed. The data is processed one DDR at a time. The processing proceeds as follows for each DDR TIM:

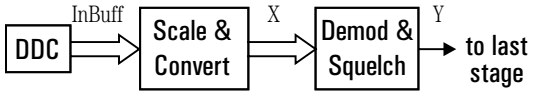
1. Check for new data from the C4x DMA and wait if none is available. If the CPU does not have to wait at this point for data from at least one of its DDR TIMs, then the processing is not real time.
2. Determine which side of the input double-buffer is current and process it through the chain shown in the general figure on page 47 and the detailed figures on pages 53, 55, and 56. In these figures, wide arrows indicate *complex* data streams and line arrows indicate *real* data streams. InBuff, X, Y, Z, and OutBuff identify the buffer holding the data at the given point in the process.
3. The integer input buffer data is converted to floating point and then scaled so that the minimum possible value on the real and imaginary sides is -1.0, and the maximum possible value is just short of +1.0. This data resides in the X buffers, segregated by channel.

Further processing steps vary depending on the modulation type. For the user type, the complex samples in the X buffers are expected to be converted into an equal number of real demodulated samples in the Y buffers. Details about the number of samples appear later in the discussion of interpolation and decimation.

Demodulation Types

The following figures show the process flow steps for each of the demodulation types. Since all types share a common end process, that flow diagram is given once.

FM Demodulation



The FM demodulator is the simplest and most efficient of the three types supplied. The algorithm employed here uses the fact that the demodulated output is proportional to the instantaneous frequency of the input, and instantaneous frequency is the time derivative of phase. Phase of the complex input (I + jQ) is arctan(Q/I), so the objective is to find:

$$\frac{d}{dt}\left\{\arctan\left(\frac{Q}{I}\right)\right\}$$

Given that

$$\frac{d}{dt}\{\arctan(u)\} \quad \text{is} \quad \left(\frac{1}{1+u^2}\right)\frac{du}{dt},$$

then

$$\frac{d}{dt}\left\{\arctan\left(\frac{Q}{I}\right)\right\} = \left(\frac{1}{1+\left(\frac{Q}{I}\right)^2}\right)\frac{d}{dt}\left\{\frac{Q}{I}\right\},$$

which simplifies to

$$\frac{|IQ'-QI'|}{I^2+Q^2},$$

where ' represents $\frac{d}{dt}\{\ \}$.

To calculate Q' and I' we use a simple 2-tap, type-4 FIR differentiator.

$$x' = x[n] - x[n-1]$$

Substituting in the numerator:

$$IQ' - QI' = I[n](Q[n] - Q[n-1]) - Q[n](I[n] - I[n-1]) \text{ simplifies to } -I[n]Q[n-1] + Q[n]I[n-1]$$

Resulting in:

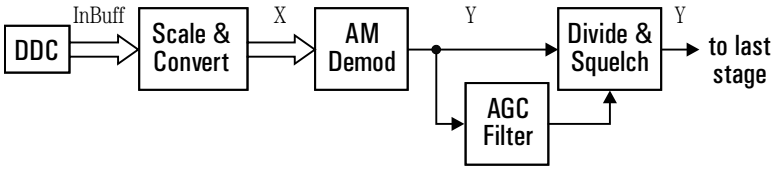
$$\text{Instantaneous Frequency} \approx \frac{1}{K}\left(\frac{-I[n]Q[n-1] + Q[n]I[n-1]}{I[n]^2 + Q[n]^2}\right)$$

Where **K** is the inverse proportionality constant:

$$K = 2\pi\left(\frac{peak\_deviation}{sample\_rate}\right)$$

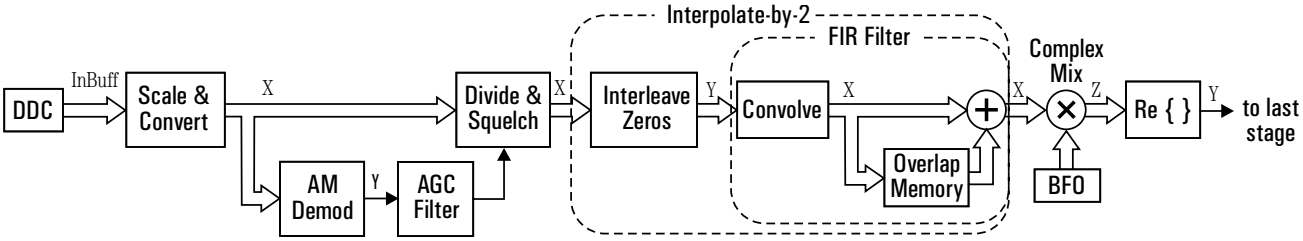
This operates on the X buffers, generating the demodulated output in the Y buffers. The magnitude-squared denominator value is compared against the squelch threshold to zero the output for too-small signals.

AM Demodulation



For AM modulation, an AM demodulator is run on the X buffers, putting the square root of the sum of the squares of the real and imaginary parts in the Y buffers. A feed-forward in-band AGC system then operates. The AGC filter runs on the data in the Y buffers. This is a single-pole no-zero IIR filter with a "diode peak-detector" to choose between attack and decay on a sample-by-sample basis. Its attack time and decay time can be set in the radio.h file, and these times remain constant despite changes in input sample rate (IF bandwidth). The demodulated signal is divided by the (relatively slowly varying) AGC filter output to generate a near-constant average output amplitude independent of input amplitude. This must be scaled safely below the clip level so that typical signals never exceed the attack time by enough to generate clipped output. The AGC filter output is also compared to the squelch threshold and the demodulated output is zeroed if the AGC filter output is too small.

SSB Demodulation

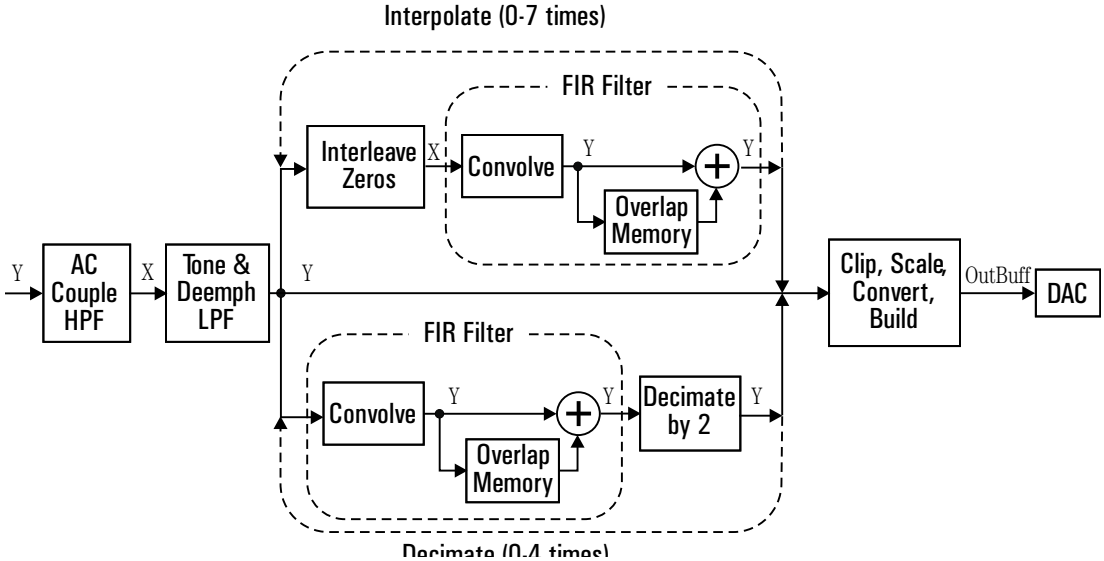


Sideband and CW demodulation is the most complicated and processing-intensive of the three demodulation types. The process begins like AM demod, except that the AGC and squelch operate on the complex input signal in the X buffers (the AGC is a preprocessing function). Next, a complex interpolate-by-two is performed (independent interpolation by two on the real and imaginary sides), using the Y buffers and ending up in X. This is necessary to prevent aliasing in the subsequent demod.

Next, the beat frequency oscillator (BFO) is applied, leaving results in the Z buffers. The BFO is implemented with a BFO-frequency-dependent phase count increment added modulo-(2^{32}) to a phase count which is then converted to a phase value. The phase is used as the argument for the sine and cosine parts of the complex BFO. To prevent aliasing, the BFO must be restricted to a range of plus or minus the IF bandwidth. The demod is completed by taking the real part, leaving the demodulated output in the Y buffers.

(This is one area where performance improvement is possible. The approach shown here is intentionally straightforward, rather than computationally optimal. Also, the BFO range restriction is a limitation for practical CW work. Complex interpolation by more than a factor of two prior to the demodulation would allow a wider BFO frequency range.)

Demod Final Stages



All types of demodulation have a common end process as illustrated in the figure above. First, a single-pole single-zero-at-DC high-pass IIR filter is applied to AC-couple the signal. Its high-pass corner frequency is set in the `radio.h` file, and, like the AGC time constant, remains constant despite changes in input sample rate. This is followed by a single-pole single-zero-at-pi low-pass IIR filter, used as a tone control (or optionally a de-emphasis filter for FM). Its corner frequency is similarly independent of input sample rate.

Home Octave

Depending on the IF bandwidth, interpolation or decimation may be required next. The audio DACs used in the DDR TIM can operate at any arbitrary sample rate within a one-octave range. For IF bandwidths which correspond to sample rates within this octave, no interpolation or decimation is required. This is also called the *home octave*. The DACs can operate at sample rates from 16.276 kHz to 32.552 kHz (see `radio.h`). For the DDCs, bandwidth is 0.55 times the (complex) output sample rate, so the home octave for IF bandwidth is from 8.95 kHz to 17.90 kHz.

For Radio, IF bandwidths outside the home octave can be used if they convert to a possible DAC rate after power-of-two interpolation or decimation, for narrower or wider IF bandwidths, respectively.

Global Variables

Three Radio program global variables related to interpolation and decimation are worth pointing out here:

- `InterpolateBy` is the interpolation factor used to bring the DDC output rate into the DAC octave.
- `RealInterpolateBy` is the interpolation factor required in the interpolation stage being discussed here. It is the same as InterpolateBy except for sideband demodulation, where it differs to account for the interpolation done in the sideband demod.
- `RealDecimateBy` describes the decimation factor required at this stage.

None of these factors are fractional. They bottom out at one if the corresponding operation is not required, and thus are always either one or a positive power of two.

Number of Samples

In the home octave, the number of samples processed is exactly the same as SAMPLES_PER_CHANNEL, defined in radio.h, everywhere throughout the process. However, if interpolation or decimation is required, things become more complicated. In the interpolation case, the C4x DMA is run at a correspondingly reduced block size so that after interpolation the data still fits the buffers. In the decimation case, decimation reduces the number of samples that are written to the output buffer from a block by the same factor that the output sample rate is smaller than the input rate.

Interpolator

The interpolation process for each power-of-two is to double the signal amplitude and interleave zeros, then filter out the images created by the zero-interleave. This is repeated as required for additional powers-of-two. The interpolation filter is designed for use in the sideband demod interpolator (the passband is pushed out slightly from what it could be for a straight interpolate-by-two design, to maximize BFO range). A 16th-order FIR filter with a +0/-1.4 dB passband response and a 70 dB stopband is used. The filter is implemented using block convolution with overlap-add to concatenate adjacent blocks.

Decimator

The decimator is similarly straightforward. For each power-of-two, the process is to filter anything beyond the bandwidth supported by the new sample rate, then throw away every other sample to reduce the sample rate by a factor of two. Repeat as required for additional powers of two. The decimation filter is only used for bandwidths beyond approximately 18 kHz (9 kHz for sideband), which are rarely used. The decimator must run as fast a possible. A 6th-order FIR filter is used. It has a 1+ dB passband, and a stopband in the low 30's of dB.

(Performance improvements are possible in the decimate and interpolate stages. Neither the interpolator nor the decimator take processing advantage of input zeros or output dead samples. Also, for multiple power-of-two sample rate changes, filters late or early in the process can be a lot less restrictive and thus faster.)

Clip, Scale, Convert, Build

The final step is to clip the output samples in the Y buffer, scale them for the 16-bit output format and convert to integer, and then build the output words by appending the identification fields to each word. These are then moved into the output buffer, where they will be handled by the DAC ISR.

Interrupt Service Routines

Timeout ISR

The timeout ISR assures continued communication with the host in the event that no EOB signal occurs for the specified amount of time.[9] Host communication occurs during the idle state and the gap ISR. The gap ISR is triggered by receiving the EOB signal. When the timeout occurs, the timeout ISR takes processing back to the idle state. The timeout time is reset and started in the start state and at the end of the GAP ISR.

The Timeout ISR can interrupt the Config state, the Process state, or the Gap ISR. When the Timeout ISR is finished, control moves to the Pre-Idle state.

DAC ISR

When the DSP processing is complete, data flows through the comm port back to DDR TIM output DACs (digital-to-analog converters). This is accomplished in a C4x interrupt service routine. The VX8 programmable IIOF2 routing matrix is configured to connect exactly one of a group's DDR TIM's sync signals to the C4x IIOF2 input. All other DDR TIM modules in a group must share the timing of this sync signal, hence the requirement of identical bandwidths across a group. In the DAC ISR, one output data sample word is written to each channel on each DDR TIM.

To assure that the DAC interrupt is *always* serviced, the other ISRs in the system must be either very short or interruptable. Radio generates a debugging signal in the DAC ISR on the C40 TIMER0 output pin (here used simply as an I/O with no timer functionality) to help you verify that the DAC interrupt is never missed. Using a logic analyzer, you can verify that every IIOF2 interrupt pulse is followed by a corresponding TIMER0 pulse.

The DAC ISR is capable of interrupting the Gap ISR.

Gap ISR

The gap ISR allows host commands to be passed to the C4x DSPs over the global shared bus when it is not being used to pass ADC data to the DDR TIMs. For more information see Passing Commands on page 50.

During the gap ISR, the following things happen in the order shown:

1. Commands from the host are read and command responses and error messages are written to the host messaging area in VX8 DRAM. The host can read and write DRAM without impacting the global shared bus due to the separation between the global shared bus and the DRAM shared bus. See page 49.
2. DDR TIMs are configured and unreset. This does not occur regularly, but rather only in the config state after start or reconfig. Unresetting the DDR TIMs in the incoming data gap ISR ensures that all DDCs receive the incoming data in a synchronized fashion after the data resumes flowing.
3. Interprocessor communication takes place concerning the desired state of the VX8 front panel user LED, which is used as an external indicator of real-time vs. non-real-time processing conditions.
4. Synchronization occurs among all C4x Radio programs guaranteeing all sites are done with global shared resources and ready for the next step.
5. Site A clears the end-of-block interrupt. This ends the gap, allowing input data to resume flowing from the system ADC.
6. The watchdog timeout timer is reset and restarted. Time-outs by the timeout timer generate the time-out interrupt, serviced by the Timeout ISR.

[9] If this occurs, the ADC FIFO has probably overflowed, resulting in a halted ADC.

Synchronization

Start and Idle Requirements

The idle and start commands require special discussion because they have system-wide implications and each require a set of synchronized actions by the host. Starting requires that the host send the start command to each group running Radio, and then start the system ADC in a timely manner. Timely means that the Radio time-out timers, running for START_TIMEOUT_CYCLES (see radio.h), don't time out. Going to idle requires sending the idle command to each group running Radio and stopping the system ADC.

Interprocessor Synchronization

One final point about the transitions among the various program states involves interprocessor synchronization. Whenever more than one group is running Radio, this is required, particularly in the gap ISR, while going idle and while starting. Interprocessor synchronization always brings with it the potential for deadlocks (one processor stuck waiting at one point, and another somewhere else). Care must be taken to avoid this. See the `share.h` and `share.c` files which contain the interprocessor synchronization implementation code.

C4x Memory Utilization

The TMS320C4x linker command files radio_a.cmd (site A) and radio_bt.cmd (all other sites) control how the linker assigns memory resources. Radio is not set up to use C4x global bus memory. This leaves C4x internal memory and C4x local bus memory available.

For Radio, the relatively small C4x internal memory is used to hold the stack. This limits the stack to 1 Kword or so and means that the fastest memory in the system is utilized by local (automatic) variables which the C compiler puts on the stack.

The .bss section is put in C40 local memory. This is still zero-wait state, but not as fast as internal memory where multiple accesses per cycle are possible in some circumstances. Larger users of memory such as the working buffers are made global or static local variables, which the C compiler puts in the .bss section.

Small Memory Model

The small memory model is used for speed and simplicity. The small model is faster than the big model because direct-addressing memory accesses count on the data-page (DP) register pointing at the .bss section, rather than reloading it every time. Its drawback is a 64 Kword limit on the size of the .bss section. The MAX_NUMBER_OF_DDRS and SAMPLES_PER_CHANNEL defined in the radio.h file affect the required size of the .bss section. After any changes to these, be sure to check the map files to be certain this .bss section size limit is not exceeded.

Stack Size

The stack is set up in the linker command files. Be aware of the size of the stack and make sure it does not overflow its allotted space. See the radio_a.cmd and radio_bt.cmd files.

Final Notes

The current version of the HSP50016 DDC chip generates a small, somewhat signal-dependent DC offset in its output when decimating by a factor which is not an exact power-of-two. This has no effect for FM or AM modulation types, but for sideband it can cause an output tone at the BFO frequency down near the noise floor. The Radio program implements a workaround involving an empirically determined DC buckout which is added to the incoming DDC signal before processing. If necessary, the buckout value can be changed with the DC buckout command. Another workaround is to operate with a power-of-two DDC decimation factor for sideband modulation. A future version of the HSP50016 DDC IC should fix this.

Keeping in mind that the Radio program is optimized as a teaching tool rather than for performance, the following table shows current approximate performance capabilities. In using this table, be aware of the home octave concept. Because of this, processing requirements are sometimes greater for a range of smaller bandwidths when additional interpolation is required, even though the time between input samples is greater.

The ability of the Radio program to operate in and out of real-time while providing real-time indication via the VX8 front panel user LED allows the user to experiment with changes in bandwidth and their effect on real-time operation.

Note

The following table is intended as an approximate guide only. The performance expressed in this table is not guaranteed!

Max. IF Bandwidth for Real-time Operation

Number of Channels	Modulation Type		
	FM	AM	Sideband
1	74 kHz	64 kHz	15 kHz
2	41 kHz	36 kHz	8 kHz
4	22 kHz	19 kHz	2.7 kHz
8	13 kHz	11 kHz	NA
12	9 kHz	NA	NA

Calculating Filter Coefficients

Finally, mention was made previously of the IF bandwidth-independent operation of the AGC filter, the AC-couple filter, and the tone-control filter. Normally when designing a digital filter, the cutoff frequency and sample rate are known, and a filter design program is used to compute the filter coefficients. For Radio, the cutoff frequencies may be specified either at compile time by changing the radio.h file (in the cases of the AGC and AC-couple filters) or at run time by the tone command (in the case of the tone-control filter). Also for Radio, the sample rate at the filter implementation point is determined at run time based on the IF bandwidth. Thus, the filter coefficients must be computed at run time.

This is done by first solving for the filter cutoff frequency in terms of its parameters, and then inverting to get the parameters in terms of the cutoff frequency. Trigonometric-series approximations are used to avoid small-angle precision problems. The end result is the required run-time filter coefficient computations in the commandInterface( ) function of the radiocmd.c module.

Building the Radio DSP Programs

To build the Radio DSP programs, you must have the following:

- TI tools
- Tartan Math Libraries
- SSVX8 Library
- HPVX8 Files
- DDRIFC Files
- Radio DSP Program Files

TI tools

You must have the TI C40 compiler/assembler/linker tools version 4.70.

Install these per TI's instructions into a single tool directory. Include the tool directory (`tool_dir`) in the system path, and set system environment variables A_DIR and C_DIR to point to the tool directory, as described in the instructions.

Recall that you must build the parallel runtime support library. Running the TI library make utility as follows in the tool directory will accomplish this:

```
mk30 -v40 -mn -o -x —h —k prts40.src
```

See the TMS320C4x Parallel Runtime Support Library Users' Guide, 1994, page 1-1.

Tartan Math Libraries

You must have the Tartan Vectar High Performance Vector Math Library for the TMS320C40, version 2.0 or greater, and the Tartan Sigtar Optimized Signal Processing Library for the TMS320C40, version 2.0 or greater. Install these per Tartan's instructions.

For Vectar, copy the vectar.h and vect40ss.lib files into the TI tool directory.

For Sigtar, make a modified copy of sigtar.h, removing the typedef for complex, and removing the declarations for the valaw and vulaw functions. These are duplicates of identical information in the vectar.h file. Without these changes, errors occur during the compile. Copy this modified sigtar.h file to the TI tool directory.

The "convl" function in the version 2.0 Sigtar library contains a bug which causes it not to work correctly with strides of two, as it is used in the Radio sideband demodulator. If your sigt40ss.lib file is dated 8/1/95, you must obtain a fix from Tartan (later library files may have incorporated this, and it is presumed that a version 2.0 library could not be dated earlier). The fix is in the form of a replacement object file for the convl function. Replace the original with the fix in a copy of the library by running the TI archiver as follows in the directory containing the copy of the library:

```
ar30 -r sigt40ss.lib convl.obj
```

Copy the fixed sigt40ss.lib file to the TI tool directory.

VX8 Library

Install the VX8 library per instructions in the *VX8 Carrier Board Installation Guide* from Spectrum Signal Processing.

The hpvx8 and ddrifc files are installed with HP VX8 Radio.

Radio DSP Program Files

This section describes the sub-directories and files contained in the \vxipnp\winXX\hpvx8\radio directory on Windows95/NT systems and the /opt/vxipnp/hpux/hpvx8/radio directory on HP-UX systems. The entire directory structures are discussed on page 9.

The following is a road map of the Radio DSP program directories and files:

```
dsp\                 # DSP (C4x) target code
   build\            # build directory
      makefile       # PC nmake makefile
      radio_a.cmd    # linker command file, site A
      radio_bt.cmd   # linker command file, all other sites
      radio_a.out    # C4x executable COFF file, site A
      radio_b.out    # C4x executable COFF file, site B
      radio_t.out    # C4x executable COFF file, TIM sites
      lst\           # listing file directory
      obj\           # object file directory
   include\          # C4x include directory
      hpinit_a.h     # C4x HP local bus setup include file
      radio.h        # C4x Radio include file
      radiocmd.h     # C4x Radio command-interface include file
      radioglb.h     # C4x Radio global include file
      share.h        # Interprocessor synchronization include file
   src\              # C4x source directory
      hpinit_a.c     # C4x HP local bus setup source file
      radio.c        # C4x Radio source file
      radiocmd.c     # C4x Radio command-interface source file
      share.c        # Interprocessor synchronization source file
host\                # host code
   include\          # host include directory
      radioifc.h     # host↔target command-interface include file
```

A PC nmake makefile is supplied. The VX8C4XSS variable near the top must be set to point to the root of the VX8 Spectrum Signal Processing library on your PC. The HPVX8_ROOT variable, also near the top, must set to point to the root of the HPVX8 software on your system. Nmake is run from the dsp build directory (the directory containing the makefile).

Appendix A: Hardware Configuration

Introduction

This Appendix details several example configurations that utilize a combination of items available from Hewlett-Packard as follows:

1. VX8 carrier board, HP SCMVX008
2. VXI shared DRAM installed in the VX8, HP SCMVX008 opts 082, 083, 085, 086
3. At least one HP DDR TIM, HP SCMVX008 opt 040
4. Optional number of C40 TIMs, HP SCMVX008 opt 011 (Spectrum MDC40SS)
5. Optional number of dual C44 TIMs, HP SCMVX008 opt 012 (Spectrum MDC44ST)

Each of the example configurations shows TIM module placement in the VX8 in addition to the appropriate jumper settings. The example configurations covered are summarized in the table below.

SCMVX008/040 Example Configurations

The following table lists 8 configurations of TIM-40 module placements as they may be delivered from the factory. Each configuration is defined by the number of different TIM module options installed and each configuration has more detail on a following page in which module locations and jumper settings are described. Also listed is whether the configuration is supported with the current version of the HP Radio demo and (if supported) the configuration name.

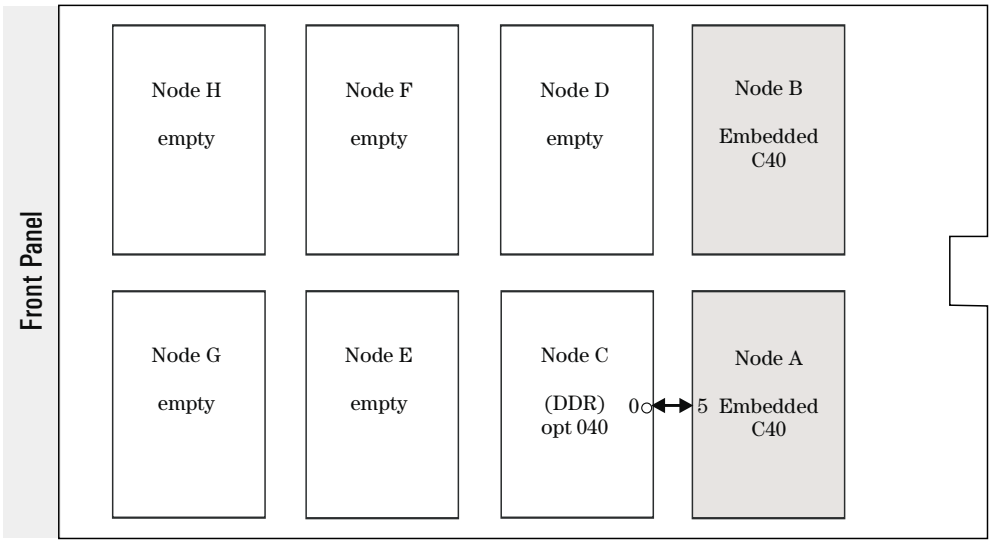
These configurations are examples. This is not an exhaustive list of possible permutations. Using the TIM-40 modules and the library software, many other configurations can be created.

Option Board Quantities			HP Radio Demo	
040(DDR)	011(C40)	012(C44)	Supported	Config
1	0	0	yes	1x1
2	0	0	yes	2x1
4	0	0	yes	2x2
6	0	0	yes	2x3
4	2	0	yes	4x1
2	2	0	no	NA
2	0	2	no	NA
4	0	2	no	NA

The example configurations are described in the pages that follow.

SCMVX008 Configuration: (1) 040, (0) 011, (0) 012

This configuration corresponds to the 1x1 HP Radio setup. See page 15.



0◀▶5 comm port connections and port numbers
○ indicates that the port is configured as an output after reset

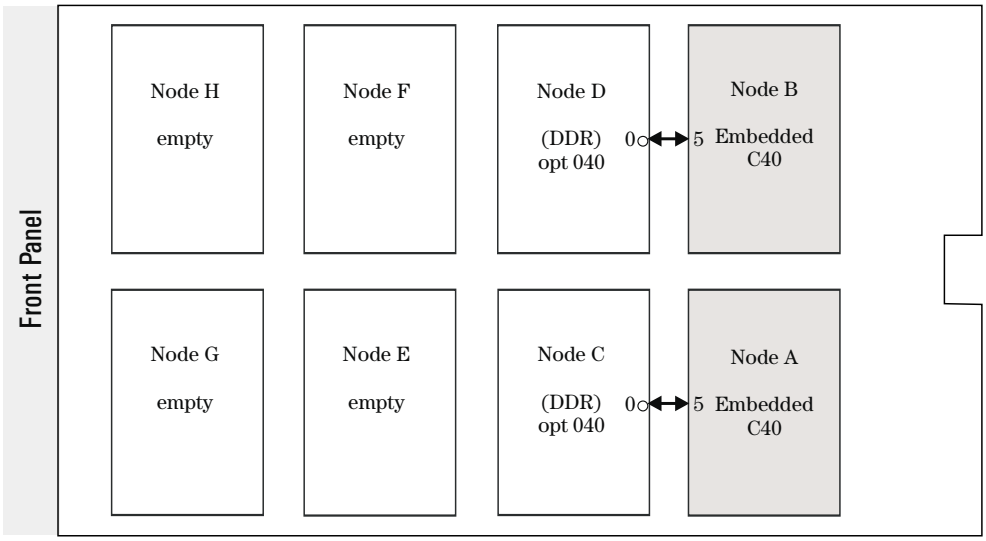
Node	TIM-40 Module Type	Jumpers			Comm Port Connections		
		040 DDR			040 DDR	C4x DSP	
		CP4	CP5	CLK	Comm Port	Node	Comm Port
C	040	out	out	out	0	A	5
D	empty						
E	empty						
F	empty						
G	empty						
H	empty						

For more information about DDR jumpers, see page 5.

For more information about VX8 comm ports, see page 6.

SCMVX008 Configuration: (2) 040, (0) 011, (0) 012

This configuration corresponds to the 2x1 HP Radio setup. See page 16.



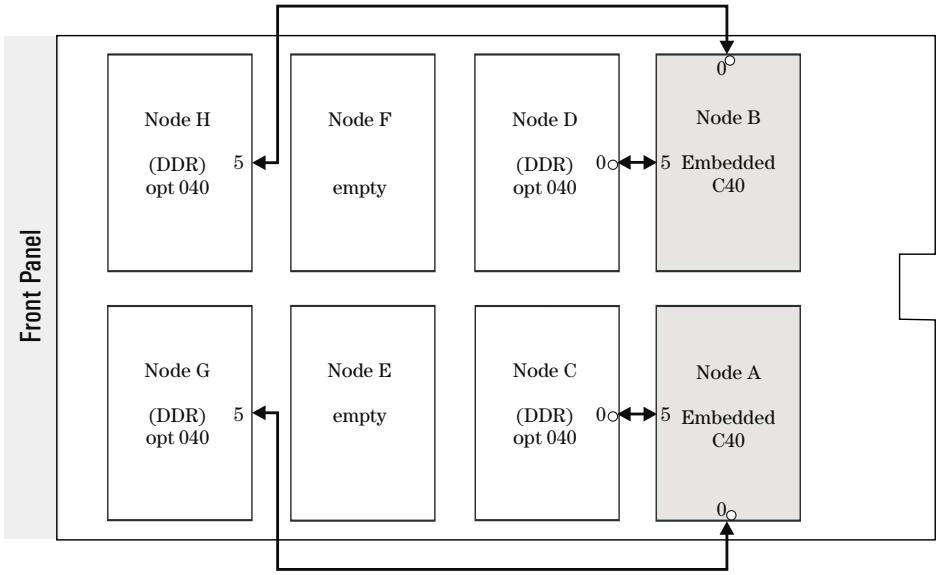
0 ◄► 5 comm port connections and port numbers
○ indicates that the port is configured as an output after reset

Node	TIM-40 Module Type	Jumpers			Comm Port Connections		
		040 DDR			040 DDR	C4x DSP	
		CP4	CP5	CLK	Comm Port	Node	Comm Port
C	040 DDR	out	out	out	0	A	5
D	040 DDR	out	out	out	0	B	5
E	empty						
F	empty						
G	empty						
H	empty						

For more information about DDR jumpers, see page 5.
For more information about VX8 comm ports, see page 6.

SCMVX008 Configuration: (4) 040, (0) 011, (0) 012

This configuration corresponds to the 2x2 HP Radio setup. See page 17.



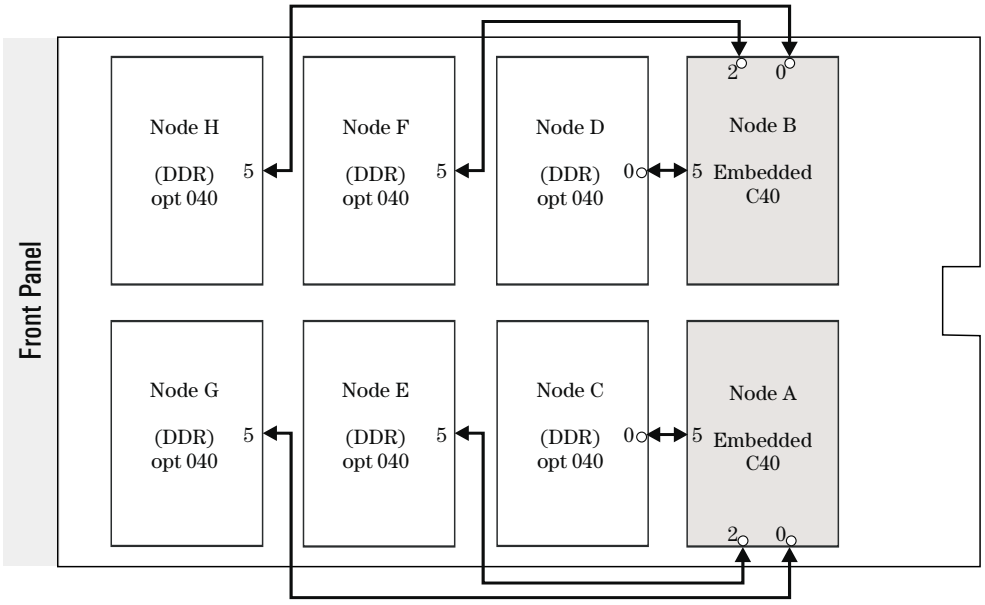
Node	TIM-40 Module Type	Jumpers			Comm Port Connections		
		040 DDR			040 DDR	C4x DSP	
		CP4	CP5	CLK	Comm Port	Node	Comm Port
C	040 DDR	out	out	out	0	A	5
D	040 DDR	out	out	out	0	B	5
E	empty						
F	empty						
G	040 DDR	out	in	out	5	A	0
H	040 DDR	out	in	out	5	B	0

For more information about DDR jumpers, see page 5.

For more information about VX8 comm ports, see page 6.

SCMVX008 Configuration: (6) 040, (0) 011, (0) 012

This configuration corresponds to the 2x3 HP Radio setup. See page 18.



0 ◄► 5 comm port connections and port numbers

○ indicates that the port is configured as an output after reset

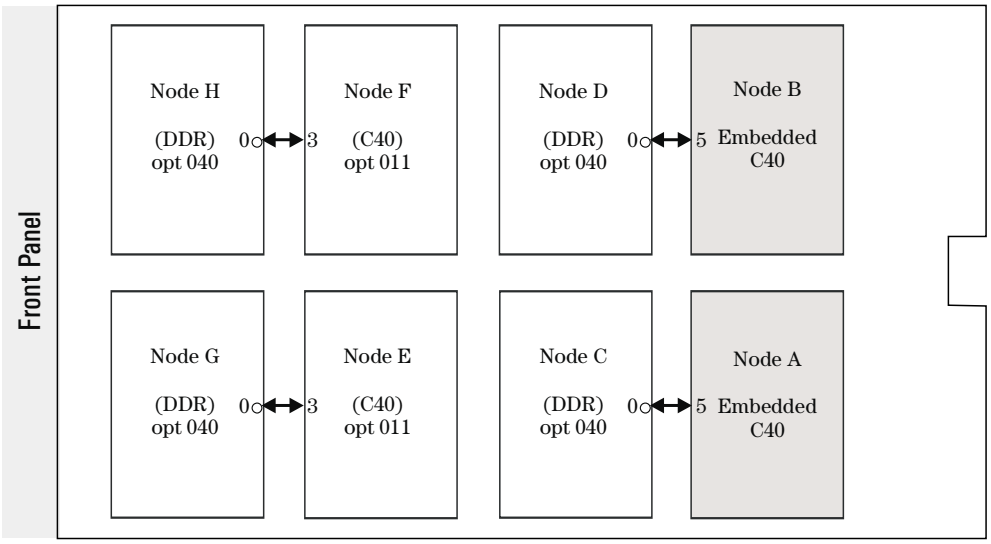
Node	TIM-40 Module Type	Jumpers			Comm Port Connections		
		040 DDR			040 DDR	C4x DSP	
		CP4	CP5	CLK	Comm Port	Node	Comm Port
C	040 DDR	out	out	out	0	A	5
D	040 DDR	out	out	out	0	B	5
E	040 DDR	out	in	out	5	A	2
F	040 DDR	out	in	out	5	B	2
G	040 DDR	out	in	out	5	A	0
H	040 DDR	out	in	out	5	B	0

For more information about DDR jumpers, see page 5.

For more information about VX8 comm ports, see page 6.

SCMVX008 Configuration: (4) 040, (2) 011, (0) 012

This configuration corresponds to the 4x1 HP Radio setup. See page 19.



0◆▶5 comm port connections and port numbers
○ indicates that the port is configured as an output after reset

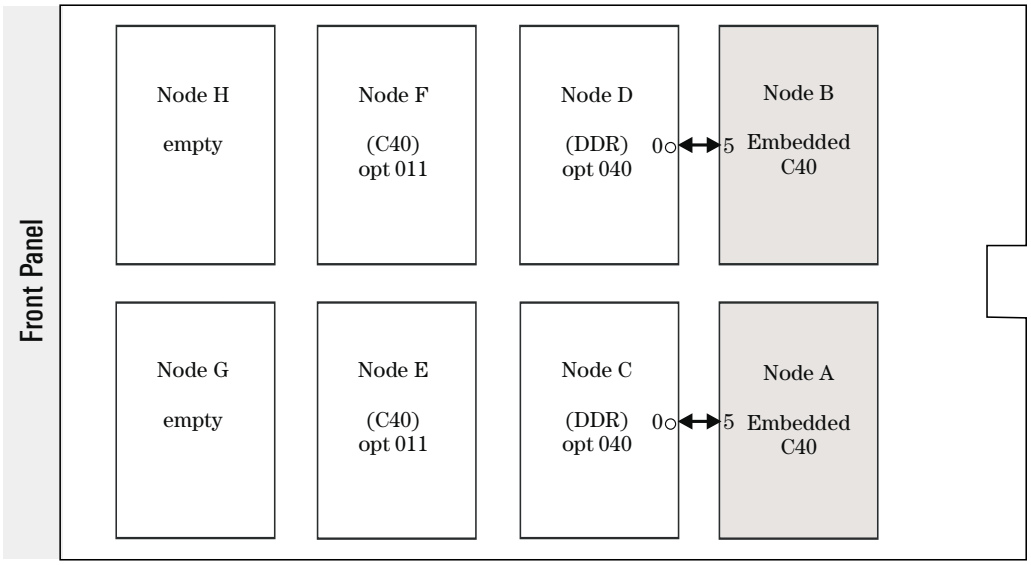
Node	TIM-40 Module Type	Jumpers						Comm Port Connections		
		040 DDR			011 C40			040 DDR	C4x DSP	
		CP4	CP5	CLK	JP1	JP2	JP3	Comm Port	Node	Comm Port
C	040 DDR	out	out	out				0	A	5
D	040 DDR	out	out	out				0	B	5
E	011 C40				out	1-2	in			
F	011 C40				out	1-2	in			
G	040 DDR	out	out	out				0	E	3
H	040 DDR	out	out	out				0	F	3

For more information about DDR jumpers, see page 5.

For more information about VX8 comm ports, see page 6.

SCMVX008 Configuration: (2) 040, (2) 011, (0) 012

This configuration is not supported by HP Radio.



0◀▶5 comm port connections and port numbers

○ indicates that the port is configured as an output after reset

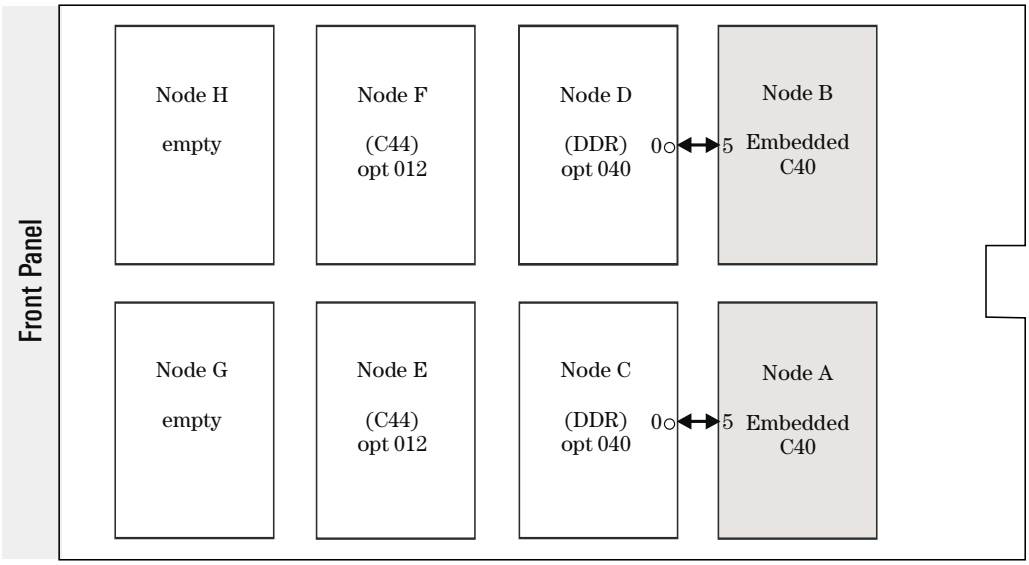
Node	TIM-40 Module Type	Jumpers						Comm Port Connections		
		040 DDR			011 C40			040 DDR	C4x DSP	
		CP4	CP5	CLK	JP1	JP2	JP3	Comm Port	Node	Comm Port
C	040 DDR	out	out	out				0	A	5
D	040 DDR	out	out	out				0	B	5
E	011 C40				out	1-2	in			
F	011 C40				out	1-2	in			
G	empty									
H	empty									

For more information about DDR jumpers, see page 5.

For more information about VX8 comm ports, see page 6.

SCMVX008 Configuration: (2) 040, (0) 011, (2) 012

This configuration is not supported by HP Radio.



0 ◄► 5 comm port connections and port numbers

○ indicates that the port is configured as an output after reset

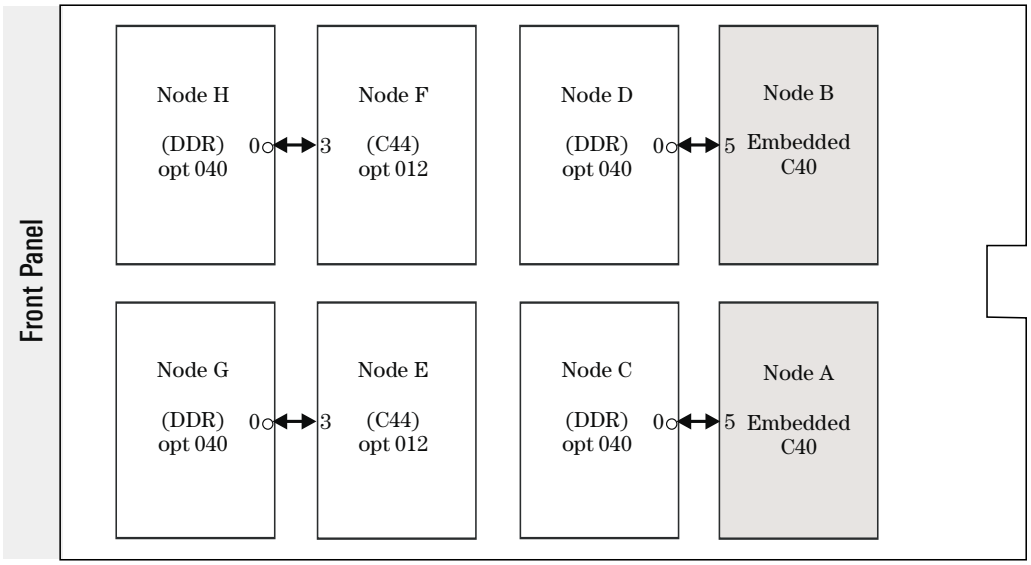
Node	TIM-40 Module Type	Jumpers								Comm Port Connections		
		040 DDR			012 C44					040 DDR	C4x DSP	
		CP4	CP5	CLK	JP1	JP2	JP3	JP4	JP5	Comm Port	Node	Comm Port
C	040 DDR	out	out	out						0	A	5
D	040 DDR	out	out	out						0	B	5
E	012 C44				1-2	out	out	out	1-2			
F	012 C44				1-2	out	out	out	1-2			
G	empty											
H	empty											

For more information about DDR jumpers, see page 5.

For more information about VX8 comm ports, see page 6.

SCMVX008 Configuration: (4) 040, (0) 011, (2) 012

This configuration is not supported by HP Radio.



0 ◄► 5 comm port connections and port numbers

○ indicates that the port is configured as an output after reset

Node	TIM-40 Module Type	Jumpers								Comm Port Connections		
		040 DDR			012 C44					040 DDR	C4x DSP	
		CP4	CP5	CLK	JP1	JP2	JP3	JP4	JP5	Comm Port	Node	Comm Port
C	040 DDR	out	out	out						0	A	5
D	040 DDR	out	out	out						0	B	5
E	012 C44				1-2	out	out	out	1-2			
F	012 C44				1-2	out	out	out	1-2			
G	040 DDR	out	out	out						0	E	3
H	040 DDR	out	out	out						0	F	3

For more information about DDR jumpers, see page 5.

For more information about VX8 comm ports, see page 6.

Index

D

E

F

G

H

S

T

U

V

W