Spectra DTP4700

Version 2.5

User Guide

PRISMTECH

Spectra DTP4700

User Guide

PrismTech

Part Number: DTP4700-UG

Doc Issue 08, 27 February 2015

Copyright Notice

Contents

Table of Contents

Preface

About the User Guide . . . . . **xi**
Contacts . . . . . **xiii**

Using Spectra DTP4700

Chapter 1 **Introduction** **3**
1.1 **Fields of Application** **3**
1.2 **System Configuration** **4**
1.3 **System Options** **4**

Chapter 2 **Overview** **5**
2.1 **System Architecture** **5**
2.1.1 Spectra DTP4700 Hardware Components 5
2.1.2 Key Features 6
2.1.2.1 Linux Operating System 6
2.1.2.2 Spectra Core Framework (CF) 7
2.1.2.3 Spectra Common Data Bus (CDB) 9
2.1.2.4 Spectra CX 10
2.1.2.5 Spectra DTP4700 Probe Toolbox 13
2.1.2.6 Hardware 14

Chapter 3 **DTP4700 BSP Overview** **15**
3.1 **Host System Configuration** **15**
3.1.1 DTP4700 Software 15
3.1.1.1 Spectra ORB (e*ORB) 15
3.1.1.2 Spectra Core Framework 16
3.1.1.3 Models for Spectra CX 16
3.1.1.4 Platform Components supplied in Binary Format 16
3.1.1.5 Utility Scripts 16
3.1.2 Host Development Environment 18
3.1.2.1 Ubuntu Host Development 18
3.1.2.2 Linux ARM Development 18
3.1.2.3 DSP Development 18
3.1.3 Network Configuration 18
3.2 **Target Filesystem** **19**
3.2.1 `platform` Directory 19
3.2.2 `applications` Directory 19

3.2.3 `bin` Directory . . . 19
3.2.4 `lib` Directory . . . 20
3.3 Thunder System Configuration. . . . 20
3.3.1 Serial Port Access . . . 20
3.3.2 Linux Kernel Configuration . . . 20
3.3.3 Linux Boot Loader. . . . 20
3.3.4 Network Configuration . . . 21
3.3.5 Root Filesystem Configuration . . . 21
3.3.6 SCA Platform Startup . . . 21
3.3.7 SCA Platform Logging . . . 22
3.4 TI Linux Terminal Services. . . . 22
3.5 DTP4700 File Transfer Services . . . 22
3.5.1 Secure Copy (SCP) . . . 22
3.5.2 SSH File Transfer Protocol (SFTP). . . . 23
3.5.3 Trivial File Transfer Protocol (TFTP). . . . 23
3.6 Additional SCA FileSystem Support . . . 24
3.6.1 Host Platform Node SCA FileSystem . . . 24
3.6.2 Spectra CX Monitor SCA FileSystem. . . . 24
3.7 Waveform FPGA Image Source Code . . . 24

Chapter 4 DTP4700 Platform 25
4.1 DTP4700 Platform Model . . . 25
4.1.1 Additional Platform Nodes . . . 27
4.2 Radio Configurations . . . 28
4.3 Development System. . . . 28
4.4 Control Scripts . . . 29
4.5 Configuring the System . . . 30
4.5.1 General Approach to Platform Configuration . . . 31
4.5.1.1 Changing Property Values in the Model . . . 31
4.5.2 Radio Frequency/Transceiver Sub-system . . . 34
4.5.2.1 The Digital Tuner . . . 37
4.5.2.2 Changing Sample Rates . . . 38

Chapter 5 Sample Waveforms 41
5.1 Restrictions on the use of the DSP . . . 41
5.2 Analog Audio (DTP4700Fm). . . . 41
5.3 Digital Audio (DTP4700FskVoice) . . . 43
5.4 Data (DTP4700FskData) . . . 45

Chapter 6 Connectivity and Communication 49
6.1 Spectra CX Build Configurations for the DTP4700 . . . 49
6.2 Modelling Platform Specific Elements . . . 50

6.3 **Connecting Waveform with Platform. 51**
6.3.1 Connecting to the RF/Transceiver Sub-System. 52

Chapter 7 **DTP4700 Probe Toolbox 59**
7.1 **DSP Probe Support. 59**
7.2 **Spectra CX Build Configurations 59**
7.3 **ProbeViz 60**
7.4 **Example Waveform 60**
7.4.1 Probe Registration 62
7.4.1.1 GPP Probe Registration 62
7.4.1.2 DSP Probe Registration 62
7.4.2 Latency Probes. 63

Chapter 8 **SCA Waveform Tutorial 65**
8.1 **The FM Waveform Model 65**
8.2 **The FM Tutorial Step-by-step. 67**
8.2.1 Starting Spectra CX 67
8.2.2 Choosing a Workspace 67
8.2.3 Importing the DTP4700 Tutorial Packages 71
8.2.4 Generating DTP4700 Platform Artefacts 73
8.2.4.1 Generating DTP4700 SCA Descriptors 75
8.2.4.2 Generating DTP4700 SCA Source Code 77
8.2.5 Building the DTP4700 Platform 79
8.2.6 Generating the FM Waveform Artefacts 80
8.2.6.1 Generating FM Waveform SCA Descriptors 81
8.2.6.2 Generating FM Waveform SCA Source Code 82
8.2.7 Building the FM Waveform. 86
8.2.8 Packaging XML and Binaries for Deployment 87
8.2.8.1 Platform 87
8.2.8.2 Waveform 88
8.2.9 Creating a DTP4700 SD Card 88
8.2.10 Running the FM Waveform. 88
8.2.10.1 Configuring Ubuntu Host Network 89
8.2.10.2 Connecting Spectra CX Monitor to DTP4700 89
8.2.10.3 Installing the FM Waveform 91
8.2.10.4 Creating the FM Waveform 93
8.2.10.5 Starting the FM Waveform. 94
8.2.10.6 Stopping the FM Waveform 97
8.2.10.7 Releasing the FM Waveform 98
8.2.10.8 Uninstalling the FM Waveform 100
8.3 **Running the FM Tutorial using cfadmin 102**
8.3.1 Starting cfadmin. 102

List of Figures

Figure 1 DTP4700 System Architecture 5
Figure 2 DTP4700 hardware components block diagram 6
Figure 3 Spectra CF Optimized Core Framework and Middleware 8
Figure 4 Spectra Common Data Bus (CDB) 10
Figure 5 Spectra CX Model-driven SCA Development System 11
Figure 6 *ProbeViz* visualization tool 14
Figure 7 Using Disk Utility 17
Figure 8 DTP4700 Platform model 25
Figure 9 Spectra CX model for x86Node 27
Figure 10 Spectra CX model for x86StandaloneNode 27
Figure 11 RfCtrlDevice component definition 32
Figure 12 Generating Descriptors 33
Figure 13 Profile descriptor 34
Figure 14 RF/Transceiver sub-system and Control Parameters 35
Figure 15 DTP4700Fm Application Assembly model 42
Figure 16 DTP4700FskVoice Application Assembly model 44
Figure 17 DTP4700FskData Application Assembly 46
Figure 18 Definition of `Demod` Component 51
Figure 19 Attaching a port to a component 53
Figure 20 Defining an Application 54
Figure 21 Uses Device Dependency 55
Figure 22 RF/Transceiver sub-system data link 57
Figure 23 ProbeViz visualization tool 60
Figure 24 DTP4700Fm Application Assembly model 61
Figure 25 Select or create a workspace 67
Figure 26 The 'Welcome' screen 68
Figure 27 Default Workbench showing empty workspace 69
Figure 28 Click the *Open Perspective* button (circled) 70
Figure 29 Open the *Spectra CX Modeling* perspective 70
Figure 30 The *Spectra CX Modeling* environment ready to start 71
Figure 31 The Import Wizard 72
Figure 32 *Project Explorer* showing imported projects 73
Figure 33 DTP4700 diagram element selected in **Project Explorer** 74
Figure 34 The DTP4700 Platform Model Element 74
Figure 35 Components in the DTP4700Node 75
Figure 36 Generating SCA Descriptors 76
Figure 37 After generating Platform Descriptors 77
Figure 38 Generating SCA Source Code 78
Figure 39 Workspace projects containing the generated files 79

Figure 40 Building the DTP4700 platform 80
Figure 41 Application Assembly Diagram for the FM waveform 81
Figure 42 Generating Descriptors 82
Figure 43 Generating C++ source code 83
Figure 44 Generating C source code 84
Figure 45 Generating All 85
Figure 46 Generated projects 86
Figure 47 Compiling and Linking 87
Figure 48 Setting a static IP address 89
Figure 49 Opening the SCA Monitor 90
Figure 50 Monitor showing running SCA platform 91
Figure 51 Installing the FM Waveform 92
Figure 52 FM Waveform successfully installed 92
Figure 53 Naming the new application instance 93
Figure 54 Successful creation of an application instance 93
Figure 55 Setting the Transmit Frequency 95
Figure 56 Choose Start from the context menu 96
Figure 57 Setting the `ptt` property 97
Figure 58 Stopping a running application instance 98
Figure 59 Releasing a stopped application instance 99
Figure 60 Application instance successfully released 100
Figure 61 Uninstalling an application 101
Figure 62 Application successfully uninstalled 102
Figure 63 `cfadmin` output confirming application has been created 103

Preface

About the User Guide

This *User Guide* provides information regarding the configuration of the DTP4700 radio hardware and SCA platform, along with guidelines for the use of the development tools and software infrastructure supporting the development of SCA-compliant waveform applications.

The *User Guide* is intended to be used *after* you have read and followed the instructions in the *Getting Started Guide*.

Intended Audience

The *User Guide* is for everyone using, developing or running SCA-compliant applications with the Spectra DTP4700.

Organisation

Chapter 1, *Introduction*, introduces Spectra DTP4700 and lists its intended end-user markets.

Chapter 2, *Overview*, describes the DTP4700 system architecture, including hardware and software components.

Chapter 3, *DTP4700 BSP Overview*, describes how to install, configure, and use the system.

Chapter 4, *DTP4700 Platform*, describes the SCA platforms provided with the system.

Chapter 5, *Sample Waveforms*, describes the example SCA waveforms provided with the system.

Chapter 6, *Connectivity and Communication*, describes the connectivity between the various components of the DTP4700 system..

Chapter 7, *DTP4700 Probe Toolbox*, describes the Probe Toolbox software and its use in the context of the DTP4700 SCA system.

Chapter 8, *SCA Waveform Tutorial*, describes step-by-step how to use the Spectra DTP4700 to generate waveforms from models, and how to run and manage the generated applications.

Conventions

The conventions listed below are used to guide and assist the reader in understanding the *User Guide*.

Item of special significance or where caution needs to be taken.

Item contains helpful hint or special information.

WIN Information applies to Windows (*e.g.* XP, 2003, Windows 7) only.

UNIX Information applies to Unix-based systems (*e.g.* Solaris) only.

C C language-specific.

C++ C++ language-specific.

Java Java language-specific.

Hypertext links are shown as *blue italic underlined*.

On-Line (PDF) versions of this document: Cross-references such as 'see *Contacts* on page xiii' act as hypertext links; click on the reference to jump to the item.

```
%  Commands or input which the user enters on the
   command line of their computer terminal
```

`Courier` fonts indicate programming code, commands, and file names.

Extended code fragments are shown in shaded boxes:

```
NameComponent newName[] = new NameComponent[1];

// set id field to "example" and kind field to an empty string
newName[0] = new NameComponent ("example", "");
```

Italics and ***Italic Bold*** are used to indicate new terms, or emphasise an item.

Sans-serif is used to indicate elements of a Graphical User Interface (GUI) or an Integrated Development Environment (IDE), such as the names of windows, panes or tabs, and the names of items displayed, such as file system directories and program objects.

Sans-serif Bold is used to indicate user actions within a GUI or IDE, such as choosing a sequence of menu options (*e.g.* **File > Save**) or clicking a **Cancel** button.

The names of keyboard keys are shown in SANS-SERIF SMALL CAPS, *e.g.* RETURN.

Step 1: One of several steps required to complete a task.

Contacts

PrismTech can be reached at the following contact points for information and technical support.

USA Corporate Headquarters

PrismTech Corporation
400 TradeCenter
Suite 5900
Woburn, MA
01801
USA

Tel: +1 781 569 5819

European Head Office

PrismTech Limited
PrismTech House
5th Avenue Business Park
Gateshead
NE11 0NG
UK

Tel: +44 (0)191 497 9900
Fax: +44 (0)191 497 9901

Web: *http://www.prismtech.com*
Technical questions: *crc@prismtech.com* (Customer Response Center)
Sales enquiries: *sales@prismtech.com*

Using Spectra DTP4700

CHAPTER

1 Introduction

The Spectra Development and Test Platform (DTP4700) is a Software Defined Radio (SDR) platform offering audio wideband, baseband (BB), and Radio Frequency (RF) capabilities. The development tools provided with the DTP4700 can be used to develop and test SCA 2.2.2 compliant systems and provide a framework for the evaluation of SCA related technology. This includes the adoption of Model Driven Design approaches to developing Software Defined Radios; specifically, the development of portable waveforms required in military, homeland security, and commercial SDR.

1.1 Fields of Application

The Spectra DTP4700 was designed with the following end-user markets in mind:

1. Waveform and application development/test teams in major radio OEMs and their end customers. Spectra DTP4700 is an ideal platform for both in-house and third-party development of SCA waveforms and applications for later deployment on target production radio platforms.
2. Advanced wireless communications (government and defense) laboratories researching areas such as cognitive radio, electronic warfare, and secure software-defined waveforms.
3. Internal research and development (IR&D) and collaborative innovation research projects into advanced wireless communications (*e.g.* SBIR, Eurostars).
4. Academic teaching and laboratory use.
5. Independent waveform and application developers creating software IP for the SDR market.

For all of these markets, Spectra DTP4700 is designed to allow SDR developers to work with the low-power OMAP 37x processor to support embedded wireless applications that are constrained by size, weight and power (SWaP).

1.2 System Configuration

Spectra DTP4700 comes pre-integrated and packaged with:

- **Thunder SDR System hardware:**
  - DM3730 OMAP based digital baseband processing system, providing GPP, DSP and FPGA processor resources.
  - Full duplex wideband RF Transceiver
  - RF Front-End to the RF Transceiver to create a high performance fully-fieldable radio system.
  - Available in either 400 MHz to 4 GHz (DTP4700H) or 30 MHz to 1.6 GHz (DTP4700L) configurations.
  - Housed in a rugged 1U enclosure with removable cover for access to the hardware.
- **Linux OS and device drivers.**
- **PrismTech's benchmark-setting SCA 2.2.2-compliant Core Framework, together with SCA Devices and ORB/CORBA Services.**
- **Demonstration SCA Waveforms/Applications.**
- **User documentation.**

1.3 System Options

The base Spectra DTP4700 package is expandable by optionally adding:

- PrismTech's Spectra CX tool for SCA component modelling, code generation, and compliance validation is the leading tool for SDR and SCA developers worldwide. Spectra CX (with the appropriate target build environment) is available as a plug-and-play upgrade for the Spectra DTP4700.
- The Spectra DTP4700 Probe Toolbox is a real-time debugging tool with the ability to prove and excite waveform elements, allowing piecemeal ('one-by-one') integration of waveform components and independent validation of temporal, processor, and memory behaviours. Probe data is visualized with the toolbox's internal visualizer, *ProbeViz*, which is available as a plug-and-play upgrade to Spectra CX.

CHAPTER

2 Overview

2.1 System Architecture

The DTP4700 System Architecture including the main hardware and software components is shown below in *Figure 1*.

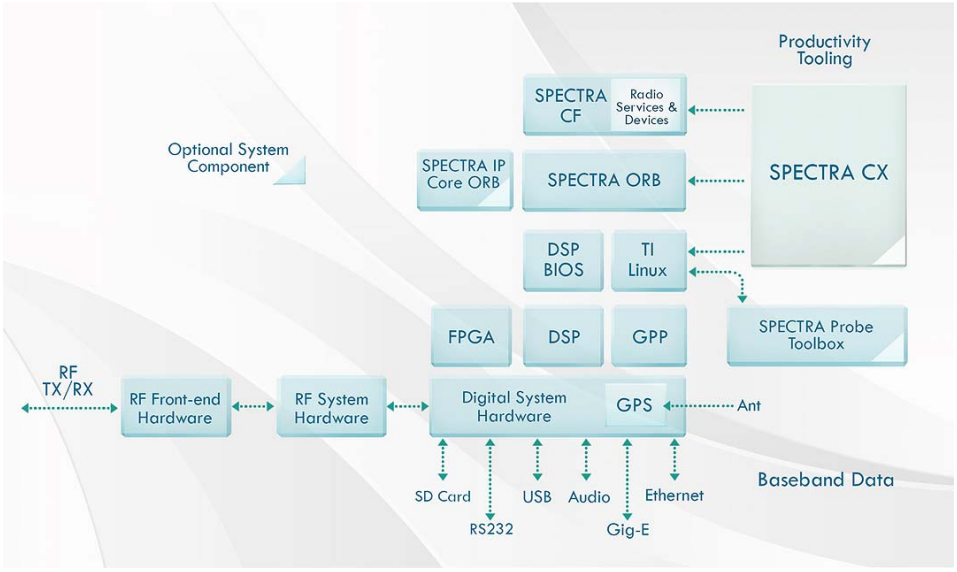

Figure 1 DTP4700 System Architecture

2.1.1 Spectra DTP4700 Hardware Components

A block diagram illustrating the main DTP4700 hardware components is shown overleaf in *Figure 2*.

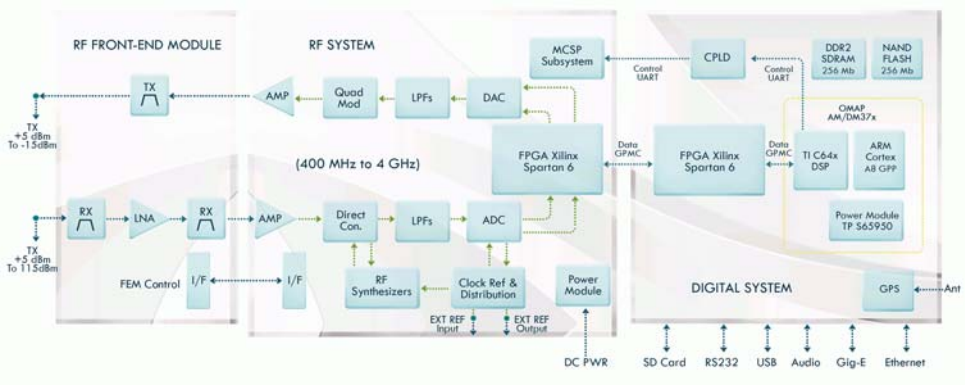

Figure 2 DTP4700 hardware components block diagram

2.1.2 Key Features

2.1.2.1 Linux Operating System

Spectra DTP4700 provides a Linux development environment based on the Digital Video Software Development Kit (DVSDK) from Texas Instruments. The DVSDK allows OMAP system integrators to quickly develop Linux-based multimedia applications that can be easily ported across different devices available in the OMAP family.

The DVSDK combines a pre-tested set of operating system, application framework, and codec libraries with example programs that demonstrate decode and encode of audio and video data streamed in real-time to/from peripheral devices. For OMAP devices that feature DSP cores, the DVSDKs provide a complete framework for developers to easily leverage DSP-accelerated codecs without having to program the DSP.

The DVSDK bundled with DTP4700 is free and does not require any run-time royalties.

DVSDK v4.03 includes the following components:

Platform Support Package

- Linux kernel 2.6.32
- Boot loaders (u-boot, x-loader)
- Linux Filesystem – Arago filesystem provides a Linux root filesystem that enable initial application development and the DVSDK demos to execute. Developers can customize the file system to their application by modifying the Arago OpenEmbedded recipes.

- TI Arago Linux SDK – provides packaged cross-building and debugging capability for OMAP/ARM.

Multimedia Package

- Multimedia Framework Product (MFP)
  - Codec Engine Framework
  - Framework Components
  - Linux Utils (CMEM)
  - XDAIS (eXpress DSP Algorithm Interoperability Standard)
- Davinci Multimedia Application Interface (DMAI)
- DSP Optimized codecs
  - Encoders: H.264, MPEG-4, JPEG, G711
  - Decoders: H.264, MPEG-4, MPEG-2, AAC, JPEG, G711
- DSP accelerated Gstreamer TI plugin

DSP Package

- C6000 code generation tool chain
- DSP/BIOS Real Time Operating System
- DSPLink Inter Processor Communication
- C6Accel – easy access to DSP accelerated function libraries
- C6Run – tool to easily run C code on the DSP

Graphics Package

- Neon accelerated Qt/Webkit application framework
- 3D Graphics Support

2.1.2.2 Spectra Core Framework (CF)

The Spectra Core Framework (CF) is a high-performance, ultra low footprint, COTS implementation of the Software Communications Architecture (SCA) standard's Framework Control and Service Interfaces and is part of PrismTech's rapidly growing family of advanced Software Defined Radio (SDR) technologies.

Spectra CF is designed specifically to support the implementation and deployment of the next-generation of complex SCA-compliant networking waveforms required for military, homeland security, and commercial SDRs.

Spectra CF can be used to support the deployment of waveform components on any mix of General Purpose Processor (GPP), Digital Signal Processor (DSP), and Field Programmable Gate Array (FPGA) processing elements.

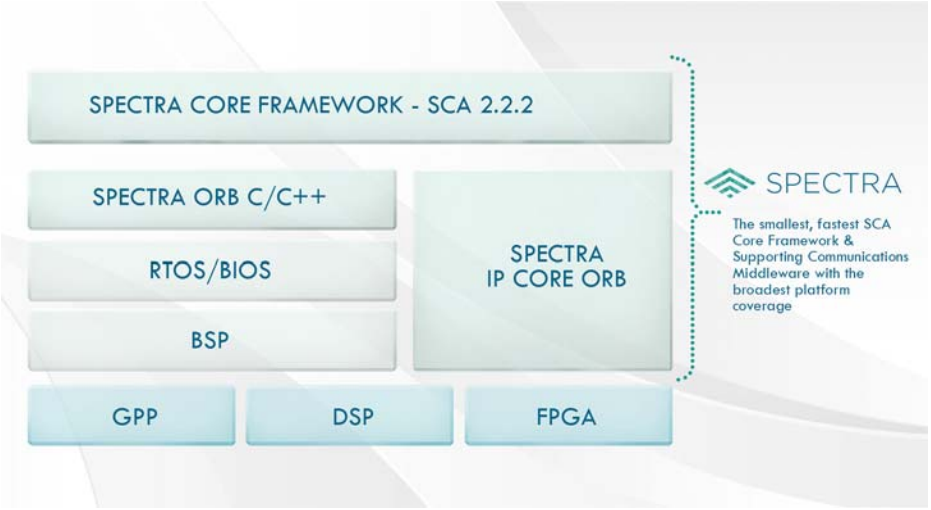

Figure 3 Spectra CF Optimized Core Framework and Middleware

Spectra CF is a fully integrated combination of SCA Framework and services implementations with the Spectra Common Data Bus (CDB) providing embedded middleware and communication transports optimized for Real-time Operating Systems (RTOS). As a supplier of the complete infrastructure PrismTech has a unique advantage in being the only company that develops both the CF and middleware components of an SCA Operating Environment (OE). This has enabled us to create a CF that has the highest performance and best Size, Weight, and Power (SWaP) characteristics available.

The C language CF has a static footprint of under 2MB, which can be as much as 10x smaller than either in-house developed or other Commercial Off-The-Shelf (COTS) CFs. The small memory footprint and optimized processing translates directly into SWaP requirements, making it suitable for ultra small form factor SDRs. The combination of advanced parsing technology, low latency middleware, and multi-threaded CF architecture enables rapid radio start-up and shutdown. Waveform applications can be started, stopped, and swapped more quickly and support a broader range of waveforms - particularly where data path latency is critical.

Spectra CF supports the deployment of both SCA platform and waveform components written in multiple programming languages, including:

- C++ (for GPPs)
- C (for GPPs and DSPs)

Spectra CF is the most efficient COTS SCA CF available, thus helping to maximize radio performance while minimizing overhead.

2.1.2.3 Spectra Common Data Bus (CDB)

Spectra Common Data Bus (CDB) is a fully integrated and optimized Software Defined Radio (SDR) Software Communications Architecture (SCA) Middleware stack.

Spectra CDB runs across a wide range of General Purpose Processor (GPP), Digital Signal Processor (DSP), and Field Programmable Gate Array (FPGA) processing elements. Spectra CDB embedded software solutions are optimized for high performance with minimal footprint on any processor choice. Spectra CDB is comprised of the following radio software infrastructure components:

- Spectra ORB (e*ORB)
  - C++ ORB (for GPP)
  - C ORB (for GPP and DSP)
- Spectra Lightweight Services
  - Spectra Lightweight Naming Service
  - Spectra Lightweight Event Service
  - Spectra Lightweight Log Service
- Spectra IP Core ORB (ICO) for FPGA and ASIC

Spectra CF and Spectra CDB embedded middleware provide the only SCA-compliant solution that is available across not only GPP, but also DSP and FPGA processing environments. This complete processor coverage has been made possible through the development of specialized CORBA middleware technology designed to support DSPs and FPGAs. PrismTech pioneered the use of lightweight ORB technology for DSPs and advanced hardware ORB technology for FPGAs. The Spectra SCA Everywhere approach helps decouple SDR applications from the underlying hardware, making hardware upgrades much more straightforward and maximizing waveform application portability.

Spectra DTP4700 includes the complete bundled CDB middleware stack with the exception of Spectra IP Core ORB which is optional.

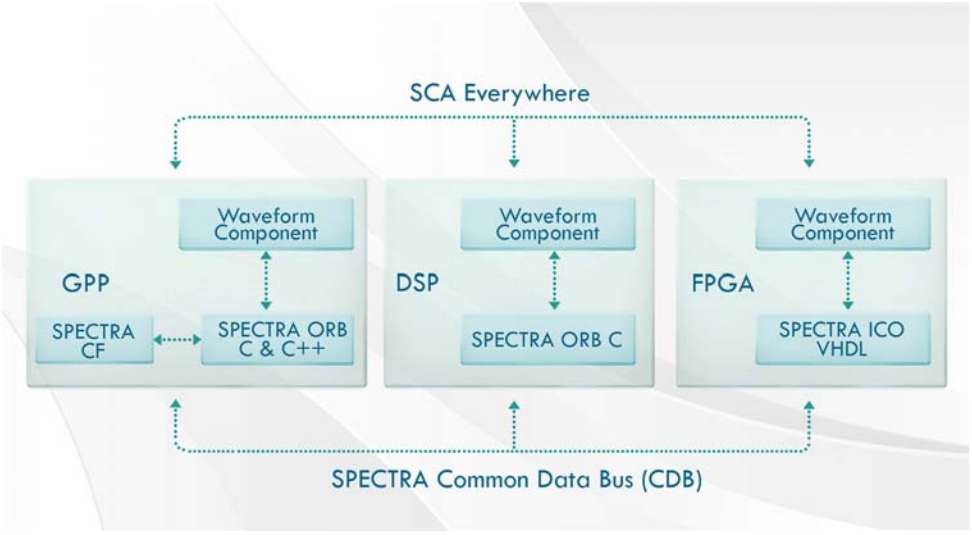

Figure 4 Spectra Common Data Bus (CDB)

Spectra CF and Spectra CDB support multiple SCA architecture options to satisfy and accelerate any platform and waveform development goals. A number of different mechanisms are provided to support communications between SDR application components running on a DSP or a FPGA. With the ability to support SCA Everywhere, the SCA's Modem Hardware Abstraction Layer (MHAL) standard, and even native communication mechanisms Spectra CF and Spectra CDB provide the SDR developer with maximum flexibility when it comes to building their SDR applications.

Spectra CDB includes a range of high-performance CORBA ORB implementations that minimize latency and memory overhead. Spectra CDB products all provide support for pluggable transports. This allows a user to choose the most efficient transport mechanism to be used for a particular processor, operating system, and radio architecture - again maximizing radio performance.

2.1.2.4 Spectra CX

Spectra CX - The SCA Development Tool

Spectra CX (SCX) is a model-driven development tool that simplifies, accelerates, and validates a significant proportion of the SCA development process. SCX validates SCA compliance at the architectural and unit test level, and generates correct-by-construction SCA-compliant artefacts, such as: XML descriptor files, compliance test reports, and validation documentation. SCX enables both SCA and non-SCA software aspects to be developed together, integrated early, and

thoroughly tested. SCX also reduces development risk due to its consistent model-based approach. Together the above benefits result in faster time-to-market, lower costs, better software quality, and superior compliance for all SCA waveform and platform code developed with SCX.

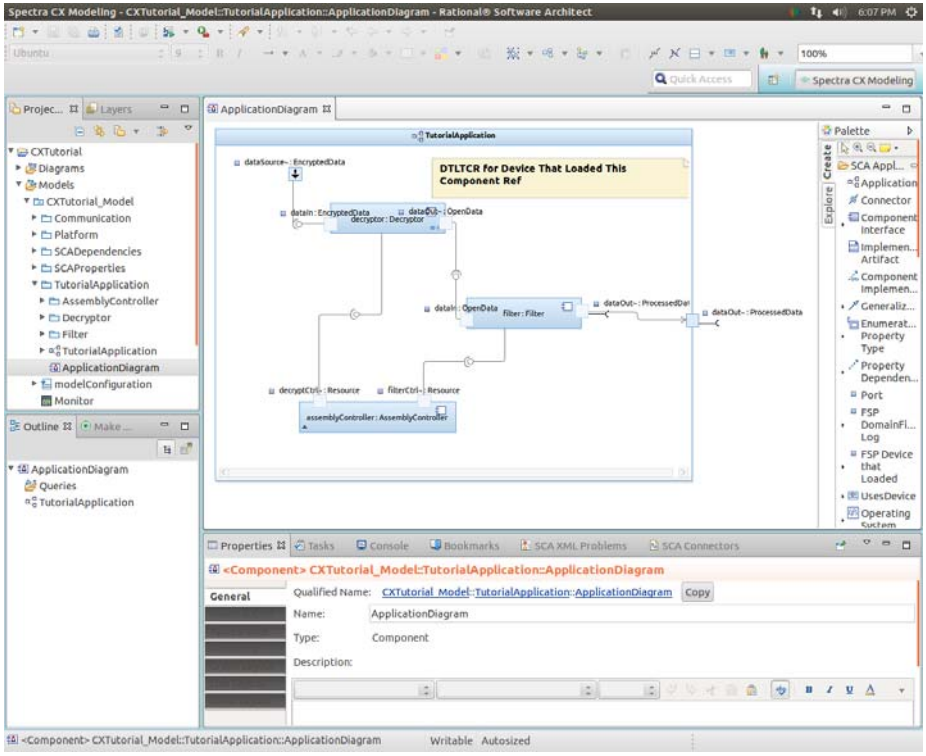

Figure 5 Spectra CX Model-driven SCA Development System

Model with Spectra CX

Benefit from complete SCA system modeling — components, assemblies, applications, nodes, platforms, deployments.

SCX provides visual modeling for all project stakeholders. Powerful visual representation of SCA concepts ensures that every project team member has a global understanding of the system and can produce correct SCA artefacts. SCX supports modeling of components, applications (waveforms), devices, platforms, and deployment of waveforms on target platforms.

Validate with Spectra CX

De-risk projects by validating early on, and throughout the development cycle.

SCX allows developers to produce SCA compliant software from day one. Validation is built right into SCX providing automatic identification of errors in SCA-compliant radio platforms and waveform applications. In addition to checking the syntax of the XML descriptors and referenced DTD's, SCX validates the semantic correctness of the model. Errors are presented together with hyperlinks to the model constructs that violate the SCA standard, and references to the relevant sections of the SCA standard.

Generate with Spectra CX

Accelerate and de-risk SCA development by generating correct-by-construction SCA artefacts

SCX provides push-button generation of correct-by-construction descriptor files and source code. Automated generation of code implementing SCA component structure is provided through Spectra's Code Generators. They automate the production of both SCA application code and SCA device code. Device code abstracts the physical hardware in accordance with the SCA specification. Automating the generation of this code benefits both developers of SCA-compliant radio platforms and developers needing to modify an SCA-compliant radio platform, allowing them to make changes to their hardware while ensuring continuous adherence to the SCA specification.

Develop with Spectra CX

Design and develop SCA component behavioral code using Model-Driven Development (MDD)

SCX provides developers with a complete model-based development environment that will significantly reduce the time to develop and maintain their components. Seamless integration with the Eclipse IDE allows developers to use their preferred tools for developing and managing source code that is linked to the model of the waveform. SCX supports the integration of behavioral models created by 3rd party UML, Block Diagram, and State Chart design tools.

Execute with Spectra CX

Monitor the component system on its actual target, with the platform running multiple applications for complete testing.

In an SCA radio, the actual deployment of software components to hardware devices is done at system initialization time. For this reason, developers require features that enable them to connect to the SCA Core Framework (CF) to

thoroughly test their application running live. In addition to features for SCA development, the SCX environment contains comprehensive features allowing developers to quickly and easily test and debug their components and applications.

SCX's runtime monitor allows users to install an application to a platform, and inspect it in real-time. With runtime monitoring, developers can see if the deployment they expected to have is actually the one dynamically created by the CF.

Test with Spectra CX

Test early and often to minimize development risk.

Automated testing of components and subsystems of an application (waveform) is provided with SCX, through the SCX SCA Test add-in (separately licensed). Generating specific code for testing the developed components for SCA compliance is critical to ensuring delivered components meet the runtime characteristics demanded by the standard. SCX SCA Test enables users to generate, compile and execute test code, and view test results directly from the tool set. Tests are executed on the Host using `JUnit` and invoke operations on a running `TargetLoader` on the target.

The Spectra CX Model Driven SCA development tool chain is an optional but strongly-recommended add-on to Spectra DTP4700. Included with Spectra DTP4700 is a 30-day evaluation copy of Spectra CX. For information on how to purchase a permanent Spectra CX license please contact your PrismTech representative.

2.1.2.5 Spectra DTP4700 Probe Toolbox

The Spectra DTP4700 Probe Toolbox is designed to reduce turn-around time for developing new waveforms on Spectra DTP4700 by providing a multi-processor debugging capability during integration.

The Probe Toolbox is a real-time debugging tool with the ability to prove and excite waveform elements, thus allowing the piecemeal integration of waveform components, and the independent validation of component temporal, processor, and memory behaviours. Probe data is visualized with the toolbox's internal visualizer, *ProbeViz*, which is integrated with Spectra CX.

The toolbox's **Data Probe** allows monitoring or injection of real-time system flow data. The **Resource Probe** provides graphical or textual representation of memory and CPU utilization and resources. The **Latency Probe** provides a graphical display of latency for user-defined probe points based on a uniform system time reference. By using the Probe Toolbox, a developer or systems engineer can thus study real-time data in any connected waveform with ease.

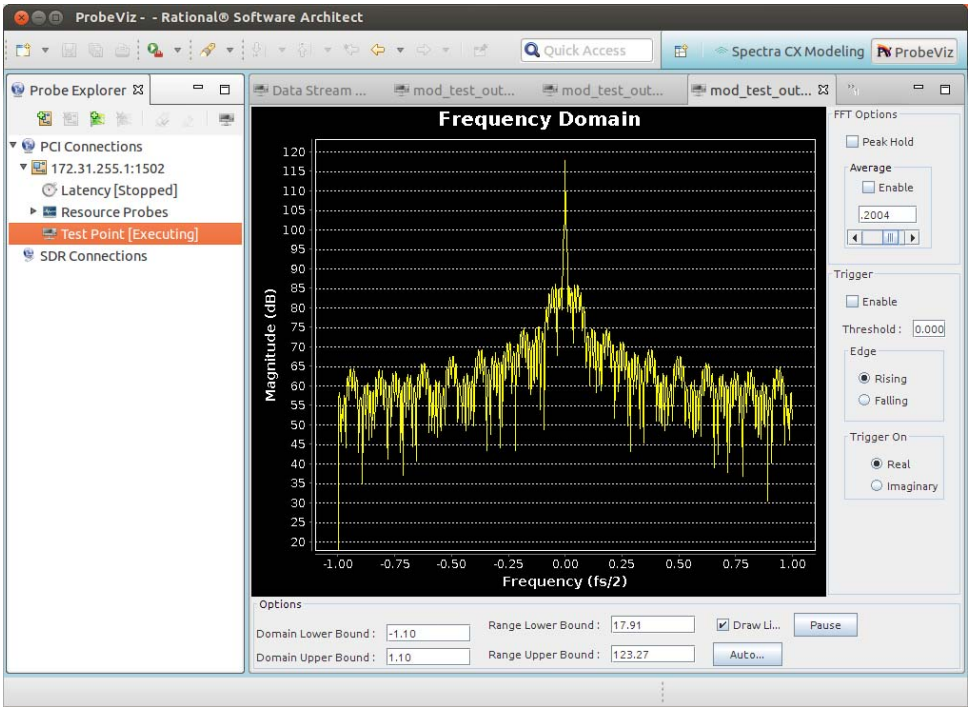

Figure 6 *ProbeViz* visualization tool

2.1.2.6 Hardware

The Spectra DTP4700 hardware platform is based on DataSoft's Thunder SDR System and consists of a highly-capable baseband Digital System assembly coupled with a wideband RF Transceiver System assembly and a Front-End Module (FEM) assembly as shown in *Figure 2* on page 6. The Digital System includes the Texas Instruments Open Media Applications Platform (OMAP) processor running an open-source Linux operating system. Numerous external interfaces provide software development and debug capabilities (GPP/DSP/FPGA). The Digital System's Waveform (WF) FPGA uses a Xilinx Spartan-6 FPGA device. The RF Transceiver System is a full-duplex design with independent RX and TX controls making it suitable for both FDD and TDD applications. The RF Transceiver design contains an RF Chain Control (RFCC) FPGA that also uses a Xilinx Spartan-6 FPGA device.

The RF Transceiver uses a generic FEM interface allowing it to host multiple FEM assemblies. The FEM assemblies provide RF filtering, input/output amplification, and RF power controls. These three electronic assemblies together create a modular system that is flexible and upgradeable.

CHAPTER

3 DTP4700 BSP Overview

The software package for the DTP4700 is supplied on a USB key and on an SD card, which contain all of the necessary software pre-installed with the operating system for the host and the target systems respectively.

The SD card operates as a bootable device for the Thunder system which bootstraps with the ARM processing unit following a power cycle.

The USB key contains the entire host-side development environment. This can boot into an Ubuntu operating system environment without writing to that computer's hard disk. You may therefore choose either to run the software directly as a Live USB key, or to install the software onto a hard disk.

Please refer to the Ubuntu installation documentation at *https://help.ubuntu.com/12.04/installation-guide/index.html*.

NOTE: If the installation instructions are not followed carefully it is possible to wipe the hard disk, as the setup routine alters the target drive's partition(s).

Alternatively, the host-side development environment can be supplied by PrismTech in the form of a Virtual Appliance instead of a Live USB key. Please contact your PrismTech representative for further details.

3.1 Host System Configuration

3.1.1 DTP4700 Software

The DTP4700 installation directory (`/opt/dtp` on the LiveUSB system) contains the DTP4700 platform and waveform artefacts, ORB and Core Framework middleware and utility scripts. Items of particular interest are described below:

3.1.1.1 Spectra ORB (e*ORB)

The DTP4700 installation directory contains a copy of Spectra ORB under the `eorb` directory which supports the following environments:

- Linux x86
- Linux ARM (Thunder)
- DSP (Thunder)

3.1.1.2 Spectra Core Framework

The DTP4700 installation directory contains a copy of the Core Framework under the `cf` directory for deployment on a Linux ARM (Thunder) target. A DSP proxy device is also provided which allows SCA Resource components to be deployed on the Thunder's DSP. An additional copy of the Core Framework for the Ubuntu host system is contained in the `host/cf` directory.

3.1.1.3 Models for Spectra CX

The Spectra CX models provided as part of the DTP4700 software, as described in Chapter *4* on page 25 and Chapter *5* on page 41 of this *Guide*, can be found in the `models.zip` archive under the DTP4700 installation directory.

Spectra CX model projects stored in zip format can be imported into a given Spectra CX workspace using the Import Existing Projects into Workspace wizard. Please refer to Section 8.2.3, *Importing the DTP4700 Tutorial Packages*, on page 71 in this *Guide* for instructions on how to access and use the import wizard.

3.1.1.4 Platform Components supplied in Binary Format

The DTP4700 installation directory contains pre-built binaries compatible with ARM for certain platform components. Specifically, these include the `AudioPortDevice`, `PacketService`, `RfCtrlDevice` and `WfFpgaProxy` components, which can be found under `target/bin`. In addition, versions containing debug symbols for the above are also included; these files are denoted by the `_g` filename suffix.

3.1.1.5 Utility Scripts

The DTP4700 installation directory contains the following utility scripts under the bin directory:

`configure-platform-version.sh` – Changes the active platform version on the host's copy of the target filesystem.

The script requires the following parameter to be provided:

`--sdk` – The install location of the TI DVSDK. On the LiveUSB system this will be `/usr/local/dvsdk`.

The platform choices are presented on the console when the script is run:

```
Select DTP Platform Version:
 1: DTP4700
 2: DTP4700 Debug
 3: Other
```

The final option, `Other`, allows a user-specific platform reference to be entered.

NOTE: This script should not be run when the host's copy of the target filesystem is being used over NFS by a Thunder target system. (See also Section 3.2.3, `bin` *Directory*, on page 19.)

`mksdboot.sh` – Creates a bootable SD card for the DTP4700 system containing a copy of the host's target filesystem.

The script requires the following parameters to be provided:

`--device` – The block device of the SD card (*e.g.* `/dev/sdd`). Tools such as Disk Utility (see *Figure 7*) can be used to determine the device.

`--sdk` – The install location of the TI DVSDK. On the LiveUSB system this will be `/usr/local/dvsdk`.

NOTE: ***This script will destroy all data on the chosen device!***

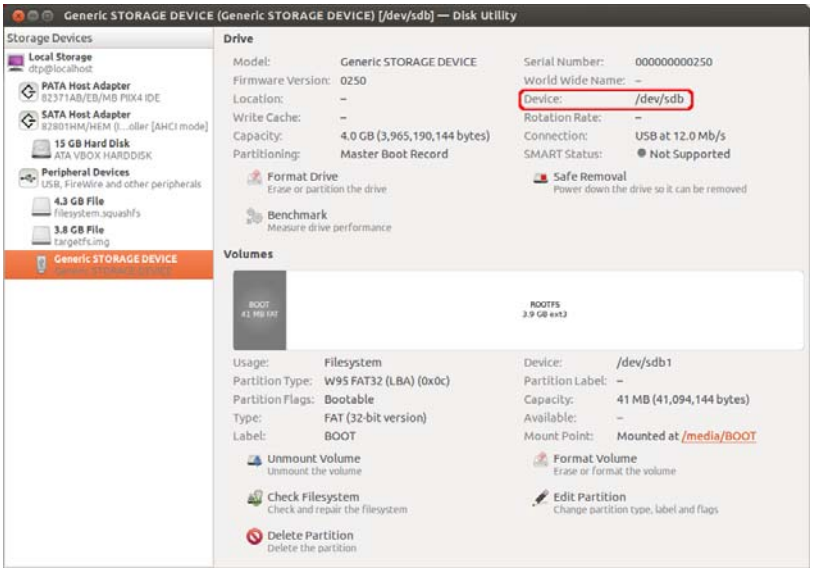

Figure 7 Using Disk Utility

Disk Utility (accessed by typing '`Disk Utility`' in the Ubuntu Dash) is used to determine the block device assigned to an SD card.

`restore-targetfs.sh` – Restores the host's target filesystem back to the original state after the DTP4700 software was installed.

The script requires the following parameter to be provided:

`--sdk` – The install location of the TI DVSDK. On the LiveUSB system this will be `/usr/local/dvsdk`.

3.1.2 Host Development Environment

The environment on the LiveUSB system is preconfigured to support development using Spectra CX for the ARM and DSP processors on the Thunder target system and the x86 processor of the Ubuntu host system. The environment is configured within the `.prismtechrc` file which is sourced by both the `.bashrc` and `.profile` files. Also, to support manual configuration of the shell environment for Spectra ORB, several scripts are provided as described below.

3.1.2.1 Ubuntu Host Development

A utility script (`host.sh`) is provided in the `bin` directory of the DTP4700 installation to configure the shell environment for developing components with e*ORB for the Ubuntu host system. To configure the shell environment, open a new shell and source the script.

3.1.2.2 Linux ARM Development

A utility script (`arm.sh`) is provided in the `bin` directory of the DTP4700 installation to configure the shell environment for developing components with e*ORB for the Thunder's ARM processor. To configure the shell environment, open a new shell and source the script.

3.1.2.3 DSP Development

A utility script (dsp.sh) is provided in the bin directory of the DTP4700 installation to configure the shell environment for developing components with e*ORB for the Thunder's DSP processor. To configure the shell environment, open a new shell and source the script.

3.1.3 Network Configuration

By default the LiveUSB system uses DHCP to configure the connection to the network. In order to establish a network connection between the Ubuntu host and Thunder systems it is necessary to configure the host system to use a static IP address.

The LiveUSB system provides two functions in the PrismTech menu to switch between DHCP and a static IP (`172.31.255.254`). These functions can be found under **Spectra > IP Address**.

The Ubuntu hosts network configuration can also be customised through the standard mechanisms provided by Ubuntu. Please refer to the Ubuntu documentation at *https://help.ubuntu.com/12.04/ubuntu-help/net.html* for further details.

3.2 Target Filesystem

During installation the TI Linux DVSDK creates a complete Linux filesystem for the target on the host system. On the LiveUSB system this filesystem can be found under the `/mnt/targetfs` directory. When creating an SD card for the target system (using the `mksdboot.sh` script) this filesystem is copied to the `ROOT` partition on the SD card. The Thunder target system can be configured to either use the root filesystem on the SD card or mount the target filesystem from the host remotely using NFS.

The DTP4700 platform and example waveform artefacts can also be found on the target filesystem under the `/opt/dtp` directory. The various subdirectories are described below.

3.2.1 `platform` Directory

The `platform` directory contains the SCA platforms available for deployment on the Thunder target system. This directory contains the binaries (in the `bin` directory) and domain profile XML for the various DTP4700 platforms. The directory also contains an xml symbolic link which links to the active SCA platform which will be started when the Thunder target system boots. Custom user platforms should be added to this directory.

WARNING: The xml symbolic link should not be manipulated manually. The appropriate `configure-platform-version.sh` script should be used to set the active platform on either the Ubuntu host or Thunder target systems.

3.2.2 `applications` Directory

The `applications` directory contains the SCA waveform applications available for deployment on the DTP4700 platform. This directory contains the DTP4700 example applications. User applications should be added to this directory.

3.2.3 `bin` Directory

The `bin` directory contains two utility scripts, `configure-platform-version.sh` and `configure-packet-service-ip.sh`.

WARNING: These scripts should *only* be run from a running Thunder target system.

The `configure-platform-version.sh` script changes the active platform version on the target.

These choices are presented on the console when the script is run:

```
Select DTP Platform Version:
 1: DTP4700
 2: DTP4700 Debug
 3: Other
```

The final option, `Other`, allows a user specific platform reference to be entered.

WARNING: If the SCA platform is running when this script is run the platform will be shut down and the new platform will be started in its place.

The `configure-packet-service-ip.sh` changes the IP address of the `PacketService` component in the DTP4700 SCA platform. The new IP address will take effect when the platform is next restarted.

3.2.4 `lib` Directory

The `lib` directory contains the Spectra ORB and Core Framework libraries required to run the DTP4700 software on the target system.

3.3 Thunder System Configuration

3.3.1 Serial Port Access

Both `minicom` and `gtkterm` are supplied on the LiveUSB system, and may be used to access the system *via* a serial cable. The following settings are required:

- Serial Device *e.g.* **`/dev/ttyS0`**
- Bps/Par/Bits:
  - **`115200`**
  - **`8N1`** (Data 8 bit, No Parity, 1 Stop bit)
- No Hardware or Software flow control.

i **NOTE:** When using the supplied USB-to-serial adapter to connect to the Thunder system the serial device name will differ from the one listed above.

3.3.2 Linux Kernel Configuration

A default Linux kernel (v2.6.32) for ARM is provided as part of the TI Linux DVSDK.

3.3.3 Linux Boot Loader

The `u-boot` boot loader is preinstalled on the Thunder SDR System. The `u-boot` command prompt can be accessed during the boot sequence by pressing any key to interrupt the boot when prompted.

During startup of the Thunder system various boot parameters are provided by the `u-boot` boot loader to control how the Thunder system boots. These values are defined as environment variables which can be changed at the `u-boot` command prompt. The following environment variables affect how the Thunder system boots:

`fsrootdev`	Device containing the root filesystem
`hostname`	Hostname of the Thunder system
`ipaddr`	IP address of the Thunder system
`netmask`	Netmask of the Thunder system
`gatewayip`	Network gateway address
`serverip`	IP address of the host system (Required for NFS)
`rootpath`	Path of the root filesystem on the host system (Required for NFS)

3.3.4 Network Configuration

By default the system is supplied set up to use a static IP address (`172.31.255.1`) to establish a network connection. The board's network configuration is configured at boot time based on parameters passed to the kernel by the `u-boot` boot loader. To modify the board's network configuration the `ipaddr`, `netmask`, and (optionally) `gatewayip` parameters should be changed in the boot loader. To obtain an IP address *via* DHCP these parameters should be left empty.

3.3.5 Root Filesystem Configuration

The Thunder system can mount the root filesystem from either an SD card or an NFS mount on the host system. By default the system is supplied set up to mount the root filesystem from the SD card. The location of the root filesystem is configured at boot time based on the value of the `fsrootdev` environment variable which should be set as follows:

- SD Card **`/dev/mmcblk0p2`**
- NFS **`/dev/nfs`**

i **NOTE**: It may also be necessary to alter the `serverip` environment variable if the IP address of the host has been changed from the default.

3.3.6 SCA Platform Startup

The TI Linux system supplied with the board should automatically start the active SCA platform. The kernel executes a startup script (`dtp.sh`) which is written for the Bourne shell and is located in the `/etc/init.d` folder on the target filesystem.

3.3.7 SCA Platform Logging

When deployed on the Thunder system any console output from the DTP4700 SCA platform is redirected to the `/var/log/dtp.log` log file. This log file will also contain any output from waveform components which have been deployed onto the ARM GPP.

3.4 TI Linux Terminal Services

TI Linux running on the Thunder system provides three different mechanisms for connecting to the system terminal. These are:

- Serial
- Telnet
- Secure Shell (SSH)

A unique system terminal is allocated for each connection. Access to the primary system terminal is *via* a serial connection to the Thunder target (see section *3.3.1* on page 20 for details). Output from the boot process will be directed to this terminal. Multiple Telnet and SSH connections can be established to the system, with each connection accessing a dedicated terminal. Once connected, the terminal behaves in the same way no matter what mechanism is used.

For more information on using Telnet or SSH please see the system manual, which can be accessed on the Ubuntu host system by typing `man telnet` or `man ssh` in the terminal.

Default login details for the system are:

```
User name: root
Password:  Not Set
```

3.5 DTP4700 File Transfer Services

The DTP4700 system provides three mechanisms for transferring files to the TI Linux filesystem. These mechanisms are described below.

3.5.1 Secure Copy (SCP)

The Secure Copy (SCP) protocol is based on the SSH protocol and provides a mechanism for securely transferring files between a local host and remote host. The host machine initiates the copy, which can be in either direction, by connecting to the remote host *via* SSH. The remote host must therefore be able to accept incoming SSH connections.

The DTP4700 system supports the copying of files *via* SCP between the Ubuntu host and the Thunder target systems. By default only TI Linux running on the Thunder target system accepts incoming SSH connections, so the transfer must be initiated from the Ubuntu host system with the Thunder system acting as the remote host.

For more information on using SCP please see the system manual, which can be accessed on the Ubuntu host system by typing `man scp` in the terminal.

3.5.2 SSH File Transfer Protocol (SFTP)

The SSH File Transfer Protocol (SFTP) is a secure file transfer protocol which, in use, is similar in function to FTP. Compared with the SCP protocol, which allows only file transfers, the SFTP protocol allows a range of operations to be invoked on the remote filesystem. For example, SFTP enables directory listings to be retrieved, and remote files can be deleted. The protocol uses SSH to establish a secure connection between a local host and remote host. For this reason the remote host must be able to accept incoming SSH connections.

The DTP4700 system supports the copying of files *via* SFTP between the Ubuntu host and the Thunder target systems. By default only TI Linux running on the Thunder target system accepts incoming SSH connections, so the connection must be initiated from the Ubuntu host system with the Thunder system acting as the remote host.

For more information on using SFTP please see the system manual, which can be accessed on the Ubuntu host system by typing `man sftp` in the terminal.

3.5.3 Trivial File Transfer Protocol (TFTP)

The Trivial File Transfer Protocol (TFTP) is a simple file transfer protocol which, compared with protocols such as FTP, provides extremely limited functionality and no authentication. While not supported by the Thunder system, a common use of TFTP is for the transfer of boot images from a host to a target system during target startup.

The DTP4700 system supports the copying of files *via* TFTP between the Ubuntu host system and the Thunder target system. The Ubuntu host system runs a TFTP server which exports the `/tftpboot` directory. Using the `tftp` command on the Thunder system, files can be retrieved from the exported directory on the Ubuntu host.

Usage of this command is:

```
tftp [OPTION]... HOST [PORT]
```

The options are:

```
-l FILE     Local FILE
-r FILE     Remote FILE
```

```
-g          Get file
-p          Put file
```

3.6 Additional SCA FileSystem Support

The DTP4700 system includes two mechanisms for binding additional, Ubuntu host based, SCA FileSystems into the DTP4700 domain. These filesystems are accessible domain-wide through the DomainManager's FileManager.

3.6.1 Host Platform Node SCA FileSystem

When bound into the DTP4700 domain the `x86Node` (described in 4.1.1, *Additional Platform Nodes*, on page 27) contributes an SCA FileSystem to the domain. The root of this filesystem is `/opt/dtp/host` on the Ubuntu filesystem. Any files added under this directory are visible across the domain.

3.6.2 Spectra CX Monitor SCA FileSystem

The Spectra CX Monitor includes a feature which allows directories on the host system to be mounted as an SCA FileSystem into the connected domain under the FileManager. Any files present under the directory on the host will be visible across the domain. In particular, directories containing generated SCA Applications can be exposed allowing them to be deployed on the DTP4700 platform without having to copy the files to the remote filesystem.

For more information on this feature, please see the Spectra CX help accessible from within Rational Software Architect by choosing **Help > Help Contents** from the main menu. The documentation for this feature can be found under **Spectra CX > Spectra CX Online Help > Runtime > Runtime Monitor** in the help system.

3.7 Waveform FPGA Image Source Code

The Digital assembly's Waveform (WF) FPGA uses a Xilinx Spartan-6 FPGA device. The WF FPGA supports digital base band processing capabilities, is user programmable and provides an interface between the RF System and the OMAP processor. The WF FPGA images can be modified and rebuilt using the Xilinx ISE tool. The VHDL source code for the Waveform FPGA image can be found under the `/usr/local/dvsdk/nimbus_sdk/fpga` directory on the Ubuntu host system.

CHAPTER

4 DTP4700 Platform

A Spectra CX model of the DTP4700 platform is included in the software package supporting the DTP4700 product. Using Spectra CX, a complete package of SCA-conformant software artefacts representing the DTP4700 platform can be generated from the model.

4.1 DTP4700 Platform Model

The domain profile for the DTP4700 platform has been pre-built and packaged such that the SCA platform will be instantiated automatically when the hardware is power-cycled. *Figure 8* shows the Spectra CX model of the DTP4700 platform.

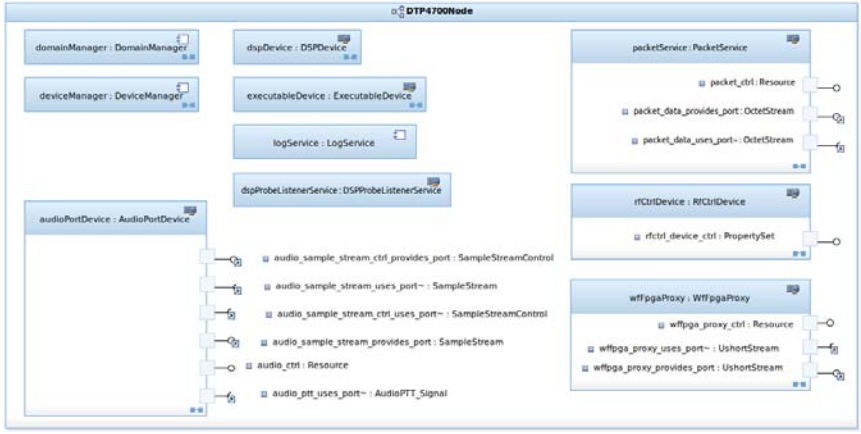

Figure 8 DTP4700 Platform model

The DTP4700 SCA platform is delivered as a single-node configuration for deployment to the ARM core of the OMAP processing unit of the Thunder hardware platform.

The DTP4700Node model element contains the following component instances:

domainManager: implements the SCA `DomainManager` interface and manages software applications, SCA Devices and DeviceManagers (nodes) within the SCA domain. The `domainManager` component instance also provides standard platform services for object location (COS Naming) and messaging (COS

Event) as part of the Spectra implementation. The `domainManager` is packaged and supplied as a COTS binary as part of the Spectra Core Framework product.

`deviceManager`: implements the SCA `DeviceManager` interface and is responsible for managing the life-cycle of logical devices and service components for the DTP4700Node node. The `deviceManager` is packaged as a COTS binary as part of the Spectra Core Framework product.

`executableDevice`: implements the SCA `ExecutableDevice` interface that supports load/unload and execute/terminate of binary-compatible application components onto the ARM. The `executableDevice` is packaged and supplied as a COTS binary as part of the Spectra Core Framework product.

`logService`: implements the OMG Lightweight Log Service specification that persists log records and retrieves log records by exposing `LogProducer` and `LogConsumer` interfaces to log service clients. The `logService` is packaged and supplied as a COTS binary as part of the Spectra Core Framework product.

`audioPortDevice`: implements the SCA `Device` interface and is conformant with the JTRS AudioPortDevice API for use by clients to send audio packets to audio output and receive audio packets from audio input. The `audioPortDevice` represents an instance of the `AudioPortDevice` component as defined in the Spectra CX DTP4700 model project and is packaged and supplied as a pre-built binary.

`rfCtrlDevice`: implements the SCA `Device` interface and provides a configuration and control interface for the Thunder transceiver hardware. The component can be used by clients to configure properties controlling output frequency, signal bandwidth, output power, *etc.*. The `rfCtrlDevice` represents an instance of the `RfCtrlDevice` component as defined in the Spectra CX DTP4700 model project and is packaged and supplied as a pre-built binary.

`wfFpgaProxy`: implements the SCA `Device` interface and provides an abstraction of the waveform (WF) FPGA API that is part of the digital system hardware. The component can be used by clients to read data from the transceiver hardware *via* the WF FPGA in receive mode and write data to the transceiver hardware *via* the WF FPGA in transmit mode. The `wfFpgaProxy` represents an instance of the `WfFpgaProxy` component as defined in the Spectra CX DTP4700 model project and is packaged and supplied as a pre-built binary.

`dspDevice`: implements the SCA `ExecutableDevice` interface and provides a proxy to the DSP processor of the DTP4700 SCA platform. The component can be used by clients to load application components onto the DSP. The `dspDevice` represents an instance of the `DSPDevice` component as defined in the Spectra CX DTP4700 model project and is packaged and supplied as a COTS binary as part of the Spectra Core Framework product.

packetService: implements the SCA `Device` interface and provides a data source/sink over the network stack of the DTP4700. The component can be used by clients to send/receive network packets to/from a virtual network interface. The `packetService` represents an instance of the `PacketService` component as defined in the Spectra CX DTP4700 model project and is packaged as a pre-built binary.

dspProbeListenerService: implements the SCA Device interface and supports the use of the Spectra DTP4700 Probe Toolbox on the DSP processor. The `dspProbeListenerService` represents an instance of the `DSPProbeListenerService` component as defined in the Spectra CX DTP4700 model project and is packaged as a pre-built binary.

4.1.1 Additional Platform Nodes

A Spectra CX model project containing additional platform node definitions is included in the software supporting the DTP4700 development system. Like the DTP4700 platform, using Spectra CX, a complete package of SCA-conformant software artefacts representing the x86/Linux operating environment available on the host can be generated from the *x86* model.

There are two node definitions included in the model project, these are shown in *Figure 9* and *Figure 10*.

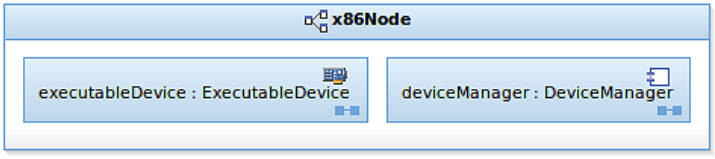

Figure 9 Spectra CX model for x86Node

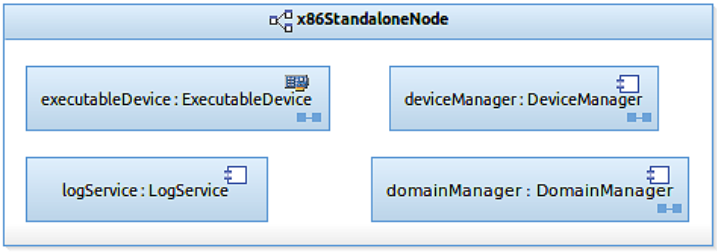

Figure 10 Spectra CX model for x86StandaloneNode

x86Node: provides an additional deployment target that can be bound into the SCA domain for the DTP4700 platform as described at the beginning of this chapter.

x86StandaloneNode: provides an independent SCA domain for the x86/Linux operating environment available on the host.

4.2 Radio Configurations

The DTP4700 product has been packaged to support separate hardware configurations of Thunder:

1. ***DTP4700H*** – with wideband RF coverage in the 400 MHz to 4 GHz frequency range.
2. ***DTP4700L*** – with RF covering the 30 MHz to 1.6 GHz range allowing users to access to the lower MILCOM frequencies.

4.3 Development System

Software products including the Spectra CX Model Driven Development System, and Spectra SCA middleware are provided to support the entire process for forward-engineering SCA-conformant radio systems on the DTP4700 hardware platform. The Board Support Packages (BSP) supporting the middleware and tools include the Linux Digital Video Software Development Kit (DVSDK), and the C6000 Code Generation toolkit from Texas Instruments. These provide the core capabilities for developing software applications to run on the OMAP 37x (Open Multimedia Applications Platform) processing unit which includes a combined general-purpose (GPP) ARM architecture and TMS320 series Digital Signal Processing (DSP) cores.

Table 1 Tools for target processors

Processor	Tools	Middleware
GPP	• Linux Digital Video Software Development Kit (DVSDK) v4.03 *Texas Instruments* • Spectra CX v3.6 *PrismTech Limited*	• Spectra ORB C and C++ Editions v2.0 *PrismTech Limited* • Spectra SCA 2.2.2 Core Framework v3.0 *PrismTech Limited*
DSP	• C6000 Code Generation Toolkit *Texas Instruments* • Spectra CX v3.6 *PrismTech Limited* • (Optional) Code Composer Studio v5 or v6 IDE *Texas Instruments*	• Spectra ORB C Edition v2.0 *PrismTech Limited*

Additionally, the baseband signal processing system on the DTP4700 includes a Spartan 6 Field Programmable Gate Array (FPGA) from Xilinx which can be programmed (optionally) using their ISE tools to meet any high speed signal processing requirements.

4.4 Control Scripts

A number of platform control scripts are provided for control of the SCA platform nodes included with the DTP4700 product. *Table 2* and *Table 3* describe the control scripts available on the Thunder target and host systems respectively.

Table 2 Thunder Target Platform Control Scripts

Script	Function
`/etc/init.d/dtp.sh`	Manages the DTP4700 SCA platform. The script is executed to start the platform when the hardware is power-cycled. The script can be used manually, and expects one of the following arguments: `start` – start the platform `stop` – stop the platform, releasing any running SCA components

Table 3 Host System Platform Control Scripts

Script	Function
`/opt/dtp/host/bin/start_platform.sh`	Starts the *x86StandaloneNode* node (platform). No arguments required.
`/opt/dtp/host/bin/stop_platform.sh`	Stops the *x86StandaloneNode* node (platform) if running. No arguments required.
`/opt/dtp/host/bin/start_node.sh`	Starts the *x86Node* and binds it into the DTP4700 platform running on the Thunder target system. This script expects the NameService reference as an argument in the format: `-ORBInitRef NameService=corbaloc:iiop:1.2@<TARGET IP Address>:2809/NameService`
`/opt/dtp/host/bin/stop_node.sh`	Stops the *x86Node* and removes it from the DTP4700 target domain. No arguments required.

4.5 Configuring the System

The adopted hardware supporting the DTP4700 SCA platform is the Thunder system from DataSoft Inc. The hardware assembly is comprised of baseband and RF/Transceiver sub-systems, and incorporates a front-end module for connection to various peripheral devices.[1]

This section describes the facilities that make-up the DTP4700 SCA platform in terms of the connectivity provided by the hardware and the configuration required to support the signal processing capabilities of the radio.

1. For detailed assembly-level and hardware design information, please refer to the documentation relating to the Thunder system provided as part of the DTP4700 product documentation bundle.

4.5.1 General Approach to Platform Configuration

The general approach to the configuration of the DTP4700 SCA platform is to alter configuration property values assigned to platform components that abstract hardware device drivers. This is entirely consistent with the SCA. There are a number of mechanisms that can be used to alter configuration property values.

- Assign property values to the component model: default values are generated from the component definition and are persisted as part of its property descriptor. Default values can be overridden at the assembly level and are persisted as part of the assembly descriptor. Descriptors are parsed by the core framework and property values are set when instantiating component instances. The procedure for setting the values of configuration properties is described in Section 4.5.1.1, *Changing Property Values in the Model*, below.
- Override configurable property values at runtime: assuming the property is writable, the value can be set by calling the configure operation of the component where the property is assigned. This is a programmatic approach that can be included as part of the application implementation itself, or made available as a general facility. The runtime monitor feature of Spectra CX is an example of the latter, where components can be interrogated at runtime, and configuration property values changed. The use of the Spectra CX runtime monitor is described in Chapter *8*, and examples of programmatically setting property values using the supplied waveforms are given for Tx and Rx baseband sample rate in Sections *4.5.2.2.1* and *4.5.2.2.2* respectively.

4.5.1.1 Changing Property Values in the Model

The general procedure for changing component default property values is described in this section. For illustration purposes, the default value of the `txFrequency` property assigned to the platform component `RfCtrlDevice` is used.

Step 1: Using Spectra CX, expand the DTP4700 model project folder in the Project Explorer view. Then expand the DTP4700 platform model folder, and the DTP4700 > RfCtrlDevice folders to display the model elements that make up the `RfCtrlDevice` component definition. Double-click the diagram element named RfCtrlDevice contained within the RfCtrlDevice package. This will display the diagram for the `RfCtrlDevice` component definition as in *Figure 11*.

Step 2: Select the txFrequency property assigned to the `RfCtrlDevice` component in the diagram. Properties are displayed on the component (structural realisation) model element as attributes.

Step 3: Open the Properties view and select the SCA tab to display the model elements associated with the txFrequency property. Change the default value as required in the field named Value. *Note*: The default value is initially set as `446006250` Hz.

Having changed the default value of the property txFrequency, the new definition for `RfCtrlDevice` can be generated as part of the SCA descriptors for the component.

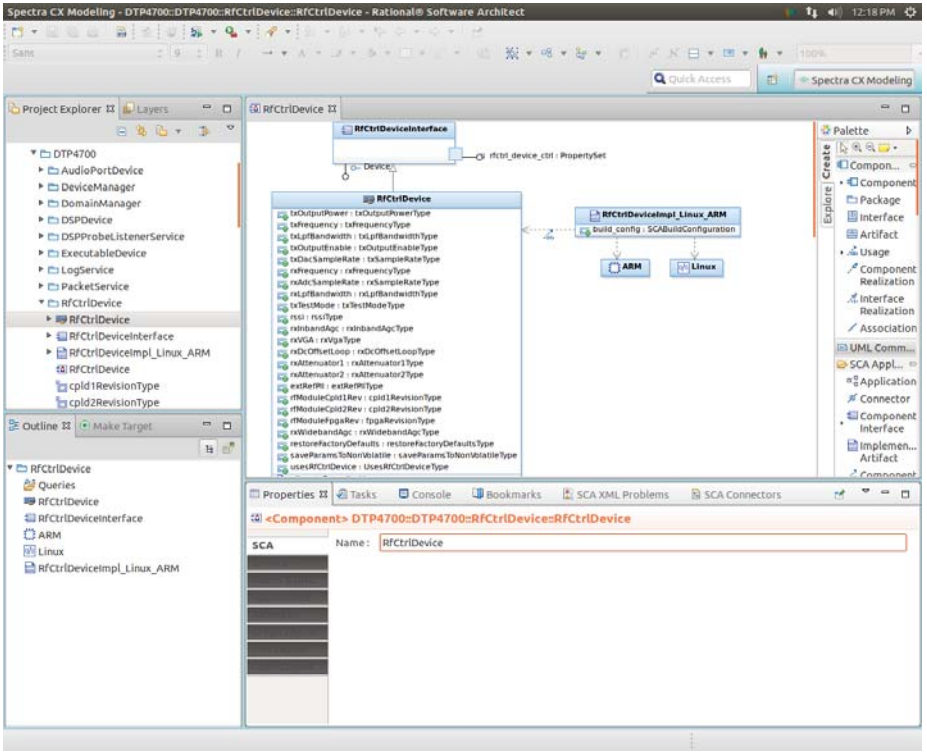

Figure 11 RfCtrlDevice component definition

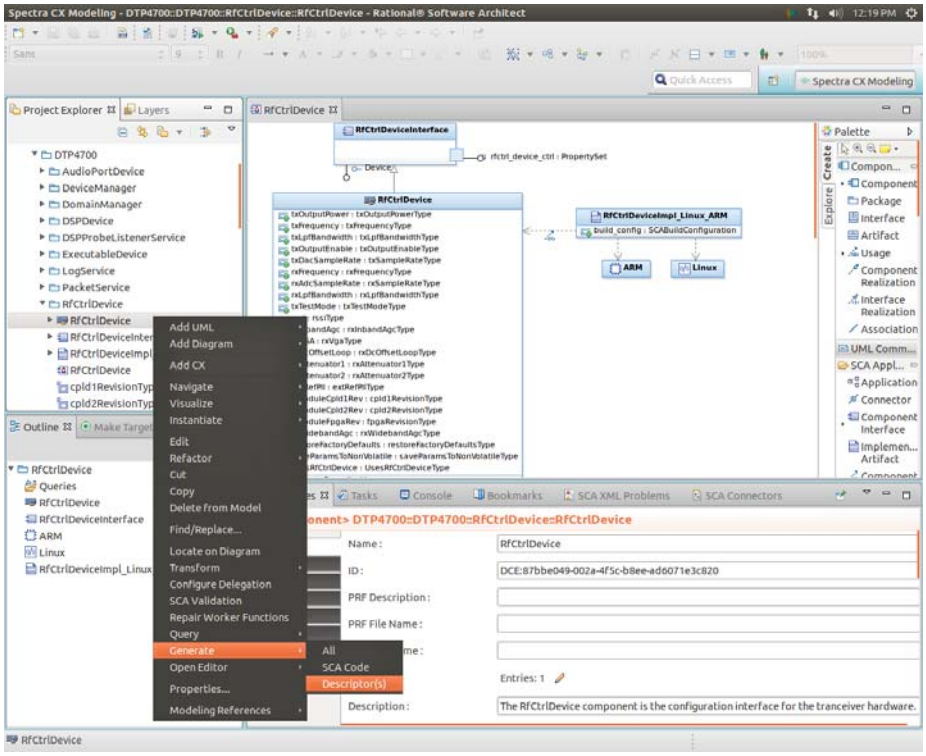

Figure 12 Generating Descriptors

Step 4: Right-click the RfCtrlDevice model element either by using the model element (structural realisation) displayed in the diagram or in the Project Explorer view, and choose **Generate > Descriptor(s)** from the menu, as shown in *Figure 12*. Notice that this procedure creates a folder `DTP4700_src` in the workspace to contain the generated descriptor artefacts, and that the model is validated prior to generating the required files.

Step 5: Expand the DTP4700_src descriptors folder in the Project Explorer view to display the generated descriptor artefacts for the DTP4700 project. Double-click on the profile descriptor for the RfCtrlDevice component (`RfCtrlDevice.prf.xml`) to view the file contents in Spectra. Using the file editor search for the text string '`txFrequency`' to examine the XML elements describing the property, including the assigned default value. This is illustrated in *Figure 13*.

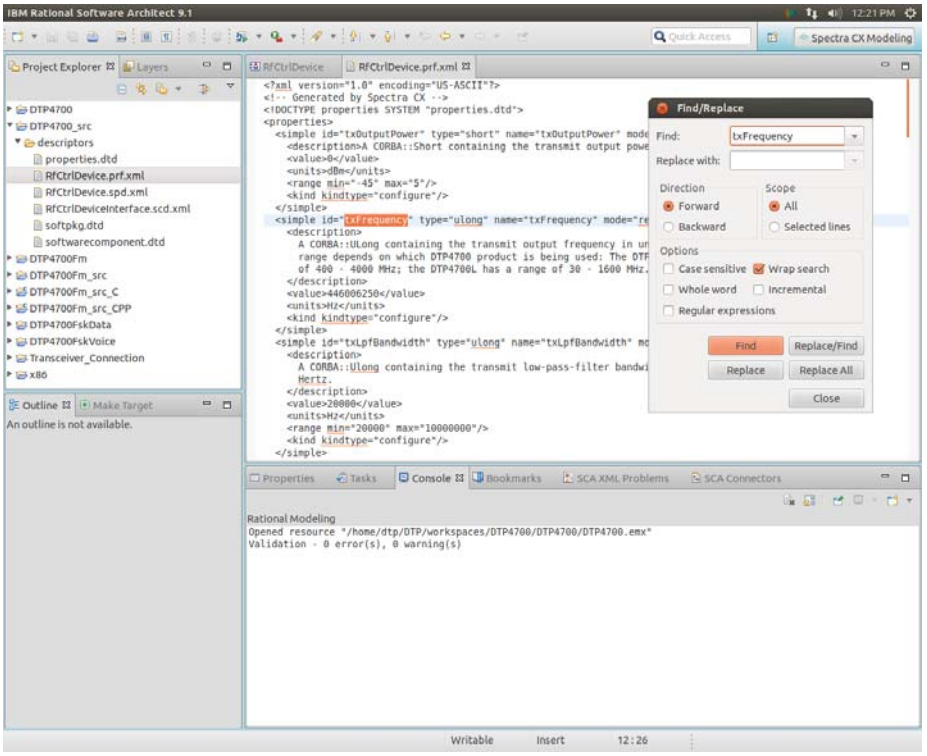

Figure 13 Profile descriptor

4.5.2 Radio Frequency/Transceiver Sub-system

The hardware components and parameters available for controlling the RF/Transceiver sub-system of the Thunder are illustrated in *Figure 14*.

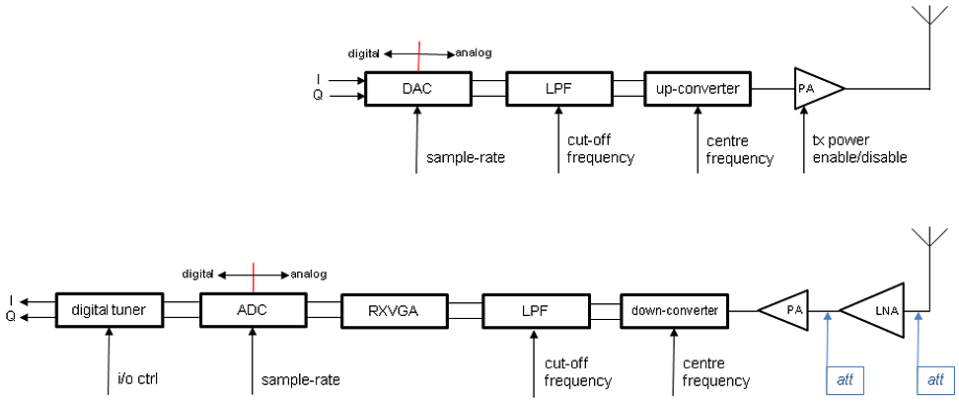

Figure 14 RF/Transceiver sub-system and Control Parameters

Referring to *Figure 14*, the software interface to the DAC (Tx) and the ADC (Rx) to the baseband domain is provided by a device driver controlling an FPGA device mounted on the digital board of the Thunder (See *Figure 2* on page 6). Control functionality for the RF/Transceiver sub-system is implemented in a separate FPGA device mounted on the RF board which is also controlled using the driver from the GPP. As described earlier in this chapter, the WfFPGAProxy and RfCtrlDevice components provide the necessary SCA abstraction for the device driver, where, in the case of data, the WfFPGAProxy is capable of sending/receiving interleaved 16-bit sampled complex baseband signal data to/from the RF/Transceiver sub-system (See Section 6.3, *Connecting Waveform with Platform*, on page 51, for more information).

The digital tuner is provided as a separate processing unit connected to the Wf FPGA. By default, signals are routed through the digital tuner. Alternatively, the digital tuner can be bypassed.

The Thunder RF/Transceiver sub-system hardware includes an Automatic Gain Control (AGC) function which supports two control loops:

- *wide-band* – affecting the Rx signal upstream of the low-pass filter (LPF) by attenuation of the signal at the Power Amplification (PA) phase (illustrated as ‘att’ in *Figure 14*).
- *narrow-band* – affecting the Rx signal downstream of the low-pass filter (LPF) by parameters controlling the Rx Variable Gain Amplifier (RXVGA).

The AGC function can be disabled in either mode which assumes maximum gain.

The various control parameters required to control the RF/Transceiver sub-system, as shown in *Figure 14*, are exposed as SCA properties assigned to the `RfCtrlDevice` component provided as part of the DTP4700 SCA platform. *Table 4* provides an overview of those properties directly affecting the RF/Transceiver sub-system control.

Table 4 SCA Properties for RfCtrlDevice Component

SCA Property Name	SCA Property Type/Action	Units/Default Value	Function/range
`txOutPutPower`	Short/Configure	dBm/0	Tx output power -45 to +5 dBm range
`txFrequency`	Ulong/Configure	Hz/446006250	Tx output frequency
`txLpfBandwidth`	Ulong/Configure	Hz/20000	Tx lowpass filter bandwidth (cut-off frequency)
`txOutputEnable`	Boolean/Configure	false	Enable Tx output
`txDacSampleRate`	Ulong/Configure	sps/48000	DAC sample rate
`rxFrequency`	Ulong/Configure	Hz/446006310	Rx input frequency
`rxAdcSampleRate`	Ulong/Configure	sps/6144000	ADC sample rate
`rxLpfBandwidth`	Ulong/Configure	Hz/20000	Rx lowpass filter bandwidth (cut-off frequency)
`rxInbandAgc`	Boolean/Configure	true	Enable automatic gain control (AGC)
`rxVGA`	Short/Configure	dB/0	Gain , only settable if rxInbandAgc is disabled /0-40
`rxDcOffsetLoop`	Boolean/Configure	true	DC offset compensation
`rxAttenuator1`	Short/Configure	dBm/31	Only settable if rxWidebandAgc is disabled
`rxAttenuator2`	Short/Configure	dBm/31	Only settable if rxWidebandAgc is disabled
`extRefPII`	Boolean/Configure	false	External ref phase lock loop
`rxWidebandAgc`	Boolean/Configure	true	Enable Rx wideband automatic gain control

4.5.2.1 The Digital Tuner

The digital tuner is used to decimate the sample rate and sharply filter the input signal, *i.e.*, reduce or eliminate aliasing caused by the ADC process[1]. By default, the digital tuner's decimation rate is 128, and the decimation filter cut-off frequency is 0.8 PI radians/sample at the decimated rate.

The decimation rate and filter cut-off can be changed by programming the digital tuner using the following method,

Step 1: Read the current configuration into memory by opening the device `/dev/ad6620`, reading into a buffer, and typecasting the buffer to `"struct AD6620_PARAMETERS_STRUCT"` (defined in `ad6620_ioctl.h`).

Step 2: The parameters can be changed as required then written back to the device.

To bypass the digital tuner, set bit 13 to `0` in Wf FPGA register 0x48. Alternatively, to route the signal through the digital tuner, set bit 13 to `1` in Wf FPGA register 0x48.

This can be achieved programmatically as follows:

```
#include <wf_fpga/wf_fpga_ioctl.h>

int fd;

fd = open("/dev/wf_fpga", O_RDWR);
if( fd < 0 )
{
   // Handle error
}

T_WF_FPGA_MEM_IO req;
req.ctrl_reg = 0x48;

if( ioctl(fd, WF_FPGA_MEM_IO_READ, &req) != 0 )
{
  // Handle error
}

// Uncomment the appropriate line.
// req.val = (req.val & ~(0x2000)); // Bypass tuner
// req.val = (req.val & ~(0x2000)) | 0x0200; // Route through tuner

if( ioctl(fd, WF_FPGA_MEM_IO_WRITE, &req) != 0 )
{
  // Handle error
}
```

1. For detailed instructions regarding the effects of setting parameters on the digital tuner please refer to the data sheet provided by Datasoft.

4.5.2.2 Changing Sample Rates

This section outlines the capabilities of the DTP4700 for tuning the sample rates of digital signals to meet various application demands.

4.5.2.2.1 Baseband Tx Sample Rate

The Tx sample rate is controlled using the `txSampleRate` configure property on the `RfCtrlDevice` (see *Table 4*).

The baseband Tx sample rate can be changed programmatically *via* the `configure` operation exposed by the `RfCtrlDevice` through its supports interface and the `rfctrl_device_ctrl` provides port. To change the rate programatically from a waveform application a connection must be made to the `rfctrl_device_ctrl` port as application components are unable to connect directly to a supports interface on an SCA Device.

As an example the `AssemblyController` component from the `DTP4700Fm` example waveform controls the sample rate from the `_CF_Resource_start` operation through its `RfCtrlDeviceCtrl` uses port which is connected to the `rfctrl_device_ctrl` provides port on the `RfCtrlDevice`. The code to change the sample rate is as follows:

```
CF::Properties p;

p.length(1);

p[0].id = "txSampleRate";
p[0].value <<= (CORBA::ULong) 48000; // samples per second

RfCtrlDeviceCtrl_->configure(p);
```

4.5.2.2.2 Baseband Rx Sample Rate

The baseband Rx sample rate is controlled using the `rxSampleRate` configure property on the `RfCtrlDevice` (see *Table 4*). However, this value must be considered in the context of the additional parameters set on the digital tuner where this is not bypassed. The actual sample rate is determined by the `rxAdcSampleRate` configure property divided by the decimation rate set on the digital tuner. In addition, the cut-off frequency set on the digital tuner is expressed in terms of *PI* radians per sample, therefore this must be adjusted to reflect the ratio of sample rate and frequency required for the application.

The baseband Rx sample rate can be changed programmatically *via* the `configure` operation exposed by the `RfCtrlDevice` through its supports interface and the `rfctrl_device_ctrl` provides port. To change the rate programatically from a

waveform application a connection must be made to the `rfctrl_device_ctrl` port as application components are unable to connect directly to a supports interface on an SCA Device.

As an example the `AssemblyController` component from the `DTP4700Fm` example waveform controls the sample rate from the `_CF_Resource_start` operation through its `RfCtrlDeviceCtrl` uses port which is connected to the `rfctrl_device_ctrl` provides port on the `RfCtrlDevice`. The code to change the sample rate is as follows:

```
CF::Properties p;

p.length(1);

p[0].id = "rxSampleRate";
p[0].value <<= (CORBA::ULong)(48000 * 128); // samples per second

RfCtrlDeviceCtrl_->configure(p);
```

4.5.2.2.3 Audio Sample Rate

The `AudioPortDevice` component supplied as part of the DTP4700 SCA platform provides an abstraction of the Advanced Linux Sound Architecture (ALSA) drivers installed as part of the Linux operating system kernel running on the ARM core of the OMAP processing unit.

The audio sample rate set on the ALSA driver is modelled as a property type (named ‘rate’) that has the SCA property kind set as `execparam`. Consequently, the core framework will interpret the `AudioPortDevice rate` property as an executable parameter for use when the component is first instantiated. The property is not configurable and its value can only be changed at the modelling (descriptor) level. This is achieved in the same way as described in Section *4.5.1* for changing the default values of configurable properties. The default value for the sample rate used by the `AudioPortDevice` is 16 kilo-samples/sec.

CHAPTER

5 Sample Waveforms

A number of Spectra CX models are included in the software package supporting the DTP4700 product. These provide sample application assemblies (waveforms) for use in platform validation and testing procedures, and for general purpose SCA training.

Using Spectra CX, the sample models can be deployed as waveform applications to the DTP4700 SCA platform as described in Chapter 8.

5.1 Restrictions on the use of the DSP

The DTP4700 does not support dynamic reconfiguration of the DSP core of the OMAP device. Consequently, a restriction is imposed whereby a single loadable image containing all the application and kernel symbols is loaded and executed only once. The approach adopted for the DTP4700, in SCA terms, is to configure a `ResourceFactory` to co-locate all components required for the DSP, and package the `ResourceFactory` together with kernel libraries as a single image using the DSP tool-chain. At the time of deployment, an SCA device running on the ARM core as part of the DTP4700 SCA platform is used as a proxy to manage the lifecycle of the DSP image. This approach is entirely conformant with the SCA.

5.2 Analog Audio (DTP4700Fm)

The functional parts of the DTP4700Fm waveform consists of four SCA components (implementation of the SCA `Resource` interface) used to transmit and receive Frequency Modulated (FM) analog audio signals over the air interface (RF) of the Thunder hardware platform. The modulation and demodulation functions are implemented in C for deployment on the DSP processor and the up-sampling and down-sampling functions are implemented in C++ for deployment on the ARM processor. *Figure 15* shows the Spectra CX model of the waveform.

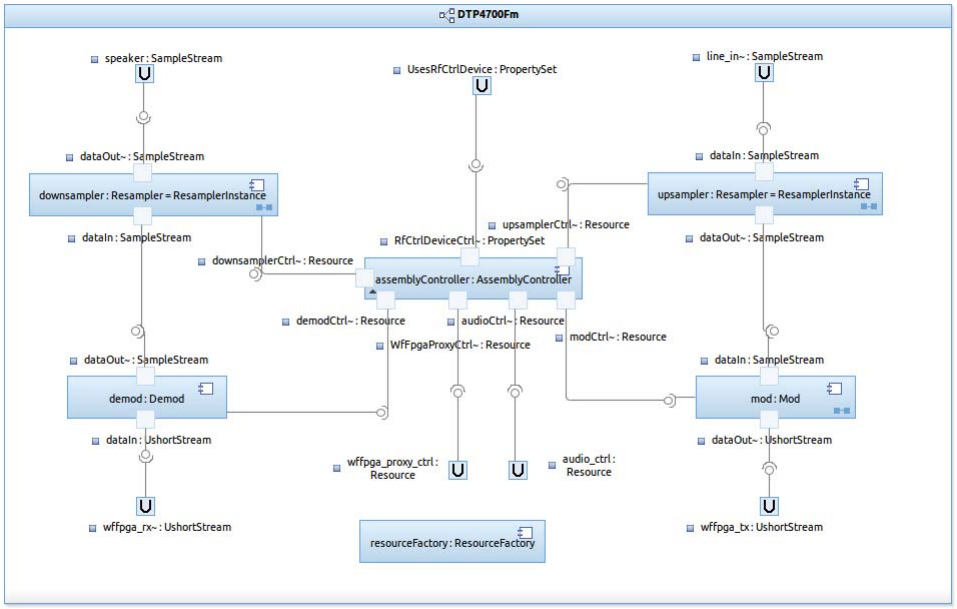

Figure 15 DTP4700Fm Application Assembly model

The DTP4700Fm application assembly illustrates the use-case where the `mod` and `demod` component instances are co-located together with a `resourceFactory` such that these can be deployed as a single binary image to the DSP.

`mod`: model element represents an instance of the `Mod` component definition. The component instance (implementation of `Resource`) receives audio samples from the `upsampler` component instance and performs a frequency modulation on the data pushing resulting complex I&Q samples to the `WfFpgaProxy` component instance running on the platform.

`demod`: model element represents an instance of the `Demod` component definition. The component instance (implementation of `Resource`) receives complex I&Q samples from the `WfFpgaProxy` component instance running on the platform and performs a frequency de-modulation on the data pushing resulting audio sample data to the `downsampler` component instance.

`upsampler`: model element represents an instance of the `Resampler` component definition that performs rate conversion. The `upsampler` instance re-samples the signal received from the `AudioPortDevice` component running on the platform from 16 to 48 kilo-samples/sec pushing resulting samples to the `mod` component instance.

downsampler: model element represents an instance of the `Resampler` component definition that performs rate conversion. The `downsampler` instance re-samples the signal received from the demod component instance from 48 to 16 kilo-samples/sec pushing resulting samples to the `AudioPortDevice` component running on the platform.

assemblyController: model element represents an instance of the `AssemblyController` component definition. Generally, the SCA requires that an assembly controller is used to delegate lifecycle operations (*e.g.*, start/stop) to connected application components using the `Resource` interface. Consequently, in the case of the Analog FM application, *uses* port connections to all other components are modelled, including, where appropriate, with platform components using a *uses device* relationship.

NOTE: The general convention is adopted for the DTP4700 models where the text '`Ctrl`' is appended to *uses* port names denoting components connected to the assembly controller. Note also that in the case where connections to platform components are required, an equivalent *provides* port is required on the platform component to represent the platform side of the resource connection.

resourceFactory: model element represents an instance of the `ResourceFactory` component definition. This provides a mechanism based on a factory design pattern for co-locating runtime instances of assigned `Resource` types with the `resourceFactory` itself. In the case of the Analog FM application, this component creates the `mod` and `demod` components.

5.3 Digital Audio (DTP4700FskVoice)

The DTP4700FskVoice waveform can be used to transmit and receive digital audio signals over the air interface (RF) of the Thunder hardware platform.

The functional parts of the waveform consist of four SCA components (implementation of SCA `Resource` interface) to provide a simple digital voice link. The encoding/modulation and decoding/demodulation functions are implemented in C for deployment on the DSP processor. *Figure 16* shows the Spectra CX model of the waveform.

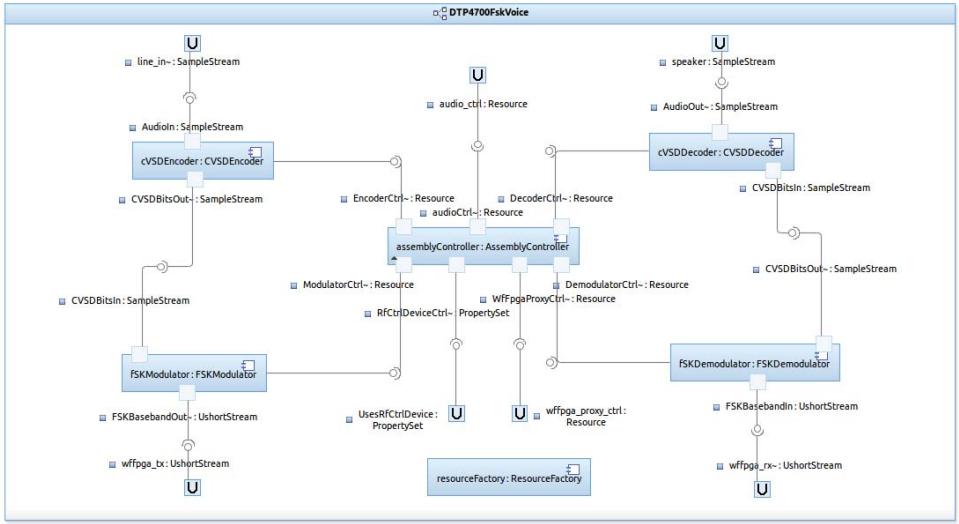

Figure 16 DTP4700FskVoice Application Assembly model

cVSDEncoder: model element represents an instance of the `CVSDEncoder` component definition. The encoder accepts a digitised voice signal from the `AudioPortDevice` component running on the platform and encodes it using Continuously Variable Slope Delta-modulation (CVSD). The voice input is 16 kilo-samples/sec, 16 bit linear quantisation. The syllabic integrator time constant is 3.5 milliseconds. The reconstruction integrator time constant is 620 us. The 16 kilo-bits/sec encoded output is pushed to the fSKModulator component instance.

fSKModulator: model element represents an instance of the `FSKModulator` component definition. The `fSKModulator` modulates a CVSD bit stream received from the `cVSDEncoder` using Frequency Shift Keying (FSK). The deviation is +/- 4 kHz, and the symbol rate is 16 kilo-symbols/sec. This form of FSK is known as Minimum Shift Keying (MSK). Each symbol comprises 4 samples. The digital I&Q baseband signal at 64 kilo-samples/sec generated output is pushed to the `WfFpgaProxy` component instance running on the platform.

fSKDemodulator: model element represents an instance of the `FSKDemodulator` component definition. The `fSKDemodulator` receives a digital I&Q baseband signal at 64 kilo-samples/sec from the `WfFpgaProxy` component instance running on the platform. Demodulation is performed by deciding whether each sample represents a positive or a negative frequency. Symbol timing recovery is

unsophisticated, and is achieved by looking for transitions in the demodulated sample stream. The output is a bit stream at 16 kilo-bits/sec, passed to the `cVSDDecoder` component instance.

`cVSDDecoder`: model element represents an instance of the `CVSDDecoder` component definition. The decoder takes the bit stream from the `fSKDemodulator` component instance and carries out CVSD decoding, using the same parameters as the encoder. The output is 16 kilo-samples/sec, 16 bit linear quantisation pushed to the `AudioPortDevice` component running on the platform.

`assemblyController`: model element represents an instance of the `AssemblyController` component definition.

`resourceFactory`: model element represents an instance of the `ResourceFactory` component definition. In the case of the Digital Audio application, this component creates the `cVSDEncoder`, `fSKModulator`, `fSKDemodulator` and `cVSDDecoder` components.

5.4 Data (DTP4700FskData)

The `DTP4700FskData` waveform can be used to transmit and receive data over the radio link of the Thunder system. The waveform makes a connection to the `PacketService` on the platform which provides the necessary data source/sink over the network stack of the DTP4700. The functional part of the waveform consists of four SCA components as shown in *Figure 17*. The deployment of the waveform has been modelled using Spectra CX such that the `modulator` and `demodulator` are installed on the DSP, and the `framer` and `deframer` are installed on the GPP of the Thunder system.

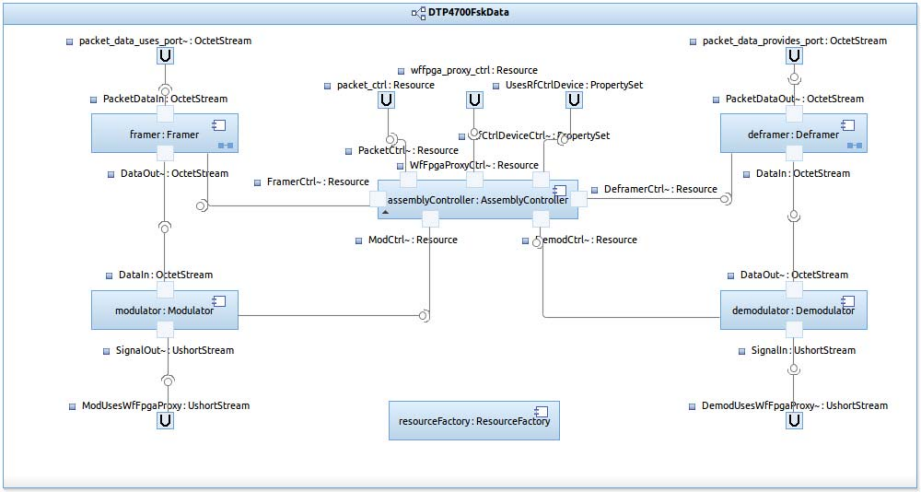

Figure 17 DTP4700FskData Application Assembly

framer: model element contained within the `DTP4700FskData` assembly represents an instance of the Framer component definition in Spectra CX. The `framer` accepts data from an instance of the `PacketService` component running on the platform. The data payload is encoded using a frame delimitation scheme based on HDLC[1]. The resulting signal is provided as input to the `modulator`.

deframer: model element contained within the `DTP4700FskData` assembly represents an instance of the `Deframer` component definition in Spectra CX. The `deframer` accepts a digital signal from an instance of the `WfFpgaProxy` component running on the platform. The input signal is decoded using a frame delimitation scheme based on HDLC. The resulting data is provided as input to an instance of the `PacketService` component running on the platform.

modulator: model element contained within the `DTP4700FskData` assembly represents an instance of the `Modulator` component definition in Spectra CX. The component modulates the input signal received from the `framer` using Frequency Shift Keying (FSK) scheme. The `modulator` produces a digital I&Q baseband signal at 64 kilo-samples/sec for the radio link by providing the signal as input to an instance of the `WfFpgaProxy` component running on the platform.

1. High-Level Data Link Control (HDLC) is a *bit-oriented* code-transparent *synchronous data link layer protocol* developed by the *International Organization for Standardization* (ISO). The frame delimitation is based on bit-stuffing, as used in HDLC for synchronous framing.

demodulator: model element contained within the `DTP4700FskData` assembly represents an instance of the `Demodulator` component definition in Spectra CX. The component demodulates the digital I&Q baseband signal received from the radio link *via* an instance of the `WfFpgaProxy` component running on the platform. The demodulated signal is provided as input to the `deframer`.

assemblyController: model element represents an instance of the `AssemblyController` component definition. The control parameters supporting the data waveform are set on the `RfCtrlDevice` by the `assemblyController` programmatically on application start-up.

resourceFactory: model element contained within the DTP4700FskData assembly represents an instance of the `ResourceFactory` component definition. The `resourceFactory` is responsible for creating instances of components of type `Modulator` and `Demodulator`.

CHAPTER

6 Connectivity and Communication

When forward engineering applications for the DTP4700 platform using Spectra CX, the connections between components are exclusively defined at the SCA modelling level.

The SCA definition includes all port connections between application components and those which connect waveform with platform. The communication model adopted for the DTP4700 platform ensures that components deployed onto the GPP and/or the DSP cores of the OMAP processing unit over such connections is facilitated using CORBA at the middleware level[1]*. The actual transport used, however, is adaptable, and is typically based on the physical interconnect fabric available. In the DTP4700, the inter-device transport used for connections made between the GPP and DSP OMAP cores is based on the Message Queue Transport (MQT) from Texas Instruments. This is also used for intra-DSP connections.*

6.1 Spectra CX Build Configurations for the DTP4700

The Spectra CX system uses a 'build configuration' to configure content and control elements of the code generation procedure applied to individual components. Principally, the build configuration is used to assign the required implementation language for a given component; however, additional features may be built-in that affect the capabilities of the generated code. Typically, therefore, the selection of a build configuration may determine the level of debugging support and/or enable the use of a particular message transport at the middleware level.

A series of build configurations have been packaged with Spectra CX to reflect the platform-specific capabilities of the DTP4700 system and the underlying Thunder hardware. *Table 5* provides a summary of the build configurations provided.

1. The term CORBA is used here to denote that interfaces between communicating components are expressed using IDL and that data is encapsulated using the GIOP message protocol.

Table 5 Spectra CX Build Configurations

Build Configuration	Target OS/ Architecture	Description
`eorb-dspbios-cl6x::eorb20-dspbios-cl6x-linux-c-dtp4700-build`	DSPBios/C6416	C based build configuration including MSGQIOP transport only
`eorb-dspbios-cl6x::eorb20-dspbios-cl6x-linux-c-dtp4700+probes-build`	DSPBios/C6416	C based build configuration including MSGQIOP transport and support for Spectra DTP4700 Probe Toolbox
`eorb20-linux-gcc-arm::eorb20-linux-gcc-arm-dtp4700+msgqiop-build`	TILinux/ARM	C++ based build configuration including MSGQIOP and IIOP transports
`eorb20-linux-gcc-arm::eorb20-linux-gcc-arm-dtp4700+msgqiop+probes-build`	TILinux/ARM	C++ based build configuration including MSGQIOP and IIOP transports as well as support for Spectra DTP4700 Probe Toolbox
`eorb20-linux-gcc-arm::eorb20-linux-gcc-arm-dtp4700-build`	TILinux/ARM	C++ based build configuration including IIOP transport only
`eorb20-linux-gcc-x86::eorb20-linux-gcc-x86-build`	Linux/x86	C++ based build configuration including IIOP transport only
`eorb20-linux-gcc-x86-c::eorb20-linux-gcc-x86-c-build`	Linux/x86	C based build configuration including IIOP transport only

Debug versions of the build configurations listed in *Table 5* are also provided.

6.2 Modelling Platform Specific Elements

Inter-device communication between the GPP and DSP on the DTP4700 platform is facilitated at the modelling level by assigning the associated component implementation model element in Spectra CX with the appropriate build configuration (see *Table 5* above and *Figure 18* on page 51).

The build configuration affects code generation for the component to which it is assigned, including the installation of the required transport to be used for the connections made at runtime between component ports. All data transports are integrated at the middleware level using the Extensible Transport Framework (ETF)

of the Spectra ORB, however, this procedure is entirely transparent to the end-user. In the case of connectivity with the DSP the native DSP/BIOS Message Queue transport mechanism from Texas Instruments is used.

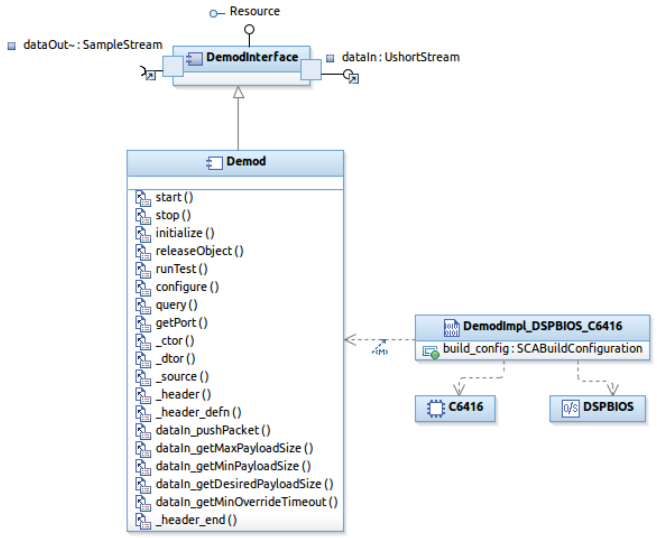

Figure 18 Definition of *Demod* Component

6.3 Connecting Waveform with Platform

There are currently three SCA platform services provided with this version of the DTP4700: one for audio, one for making connections to the network interface, and one for the radio link (RF/Transceiver sub-system). These are provided by the `AudioPortDevice`, the `PacketService`, and by a combination of the `RfCtrlDevice` and `WfFpgaProxy` components supporting the radio link respectively. The latter provides an SCA abstraction of the low-level Wf FPGA device driver libraries allowing configuration and control of the Thunder RF/Transceiver sub-system hardware, and a connection for exchanging sampled complex baseband signal data for the RF transmit and receive paths.

This section of the *User Guide* provides guidance for modelling connections between application components and DTP4700 platform components. In general waveform components (`CF::Resource`) connect to the platform by specifying uses device dependencies based on allocation properties defined by the DTP4700 platform. *Table 6* shows the allocation properties defined by the various DTP4700 platform components and the value required to create a connection.

Table 6 DTP4700 Platform Allocation Properties

Device	Property Name	Required Value
AudioPortDevice	UsesAudioPortDeviceType	1
PacketService	UsesPacketServiceType	1
RfCtrlDevice	UsesRfCtrlDeviceType	1
WfFpgaProxy	UsesWfFpgaProxyType	1

The example provided in the next section demonstrates how to create a waveform component (`CF::Resource`) in Spectra CX with connections to transfer signal data to and from the RF/Transceiver sub-system of the Thunder system.

6.3.1 Connecting to the RF/Transceiver Sub-System

Step 1: Using Spectra CX, right-click in the Project Explorer view and choose **New > SCA Model Project** from the menu. Name the project `Transceiver_Connection`, then click **Next**. Name the default SCA model `Transceiver_Connection`, and then click **Finish**.

Step 2: The Rational model editor should open automatically for the new model. With the editor visible, switch to the Details tab and in the Model Libraries section click the **Add** button. Select JTRS Packet API from the Deployed Library drop-down list and click **OK**. Save the model.

Step 3: Expand the Transceiver_Connection project in the Project Explorer view, right-click the Transceiver_Connection model and choose **Add CX > Component** from the menu. Name the component `Modem`, ensuring that the component type is set to `Resource`, then click **Finish**.

Step 4: The diagram for the `Modem` component will open automatically. With the diagram visible, and with the Port object under SCA Application drawer of the Palette view selected, left-click on the `ModemInterface` model element in the diagram to attach a port to the `Modem`.

Step 5: Select the port element in the diagram and open the Properties view to display the model properties for the port. Ensure that the SCA tab in the Properties view is selected. In the Properties view, give the port the name `dataOut`, and set the Conjugation to `Uses`. Click the pencil icon in the Properties view to define the Porttype, and in the resulting dialog box type `UshortStream` in the Choose an element field. Select the UshortStream - Packet::Packet::Packet::UshortStream interface type from the resulting list of available interfaces, then click **OK**.

Step 6: Repeat Steps 4 and 5 to create another port, this time give the port the name `dataIn`, and set the Conjugation to `Provides`.

This stage is illustrated in *Figure 19*.

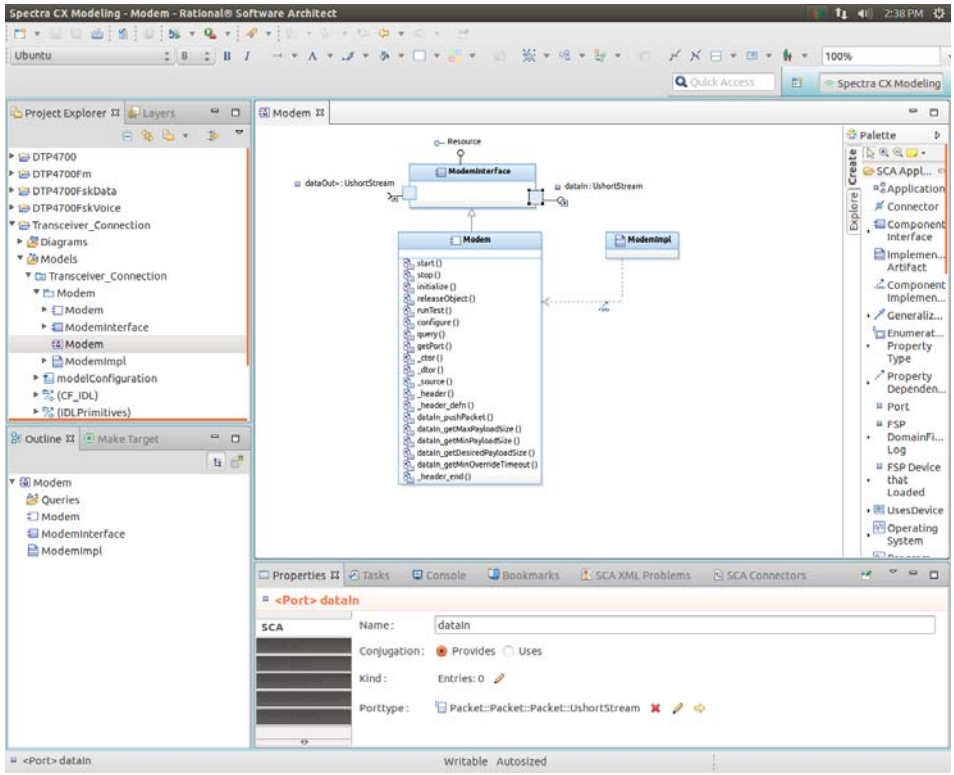

Figure 19 Attaching a port to a component

Step 7: Right-click on the Transceiver_Connection model in the Project Explorer view and select Add **Diagram > CX Component Diagram** from the menu. Name the diagram `Transceiver_Connection`. With the diagram visible, and with the Application object under SCA Application drawer of the Palette view selected, left-click on the diagram canvas to create an application model element. Name the application `Transceiver_Connection`.

Step 8: With the Transceiver_Connection diagram containing the Transceiver_Connection application model element visible, expand the Modem model folder in the Project Explorer view. Left-click on the on the Modem model element (structural realization) and drag the icon onto the diagram canvas to a location inside the bounding box of the Transceiver_Connection application. This will include an instance of the `Modem` component definition into the Transceiver_Connection

application. A corresponding model element for the Transceiver_Connection application will appear in the Project Explorer view under the Transceiver_Connection model. This is illustrated in *Figure 20*.

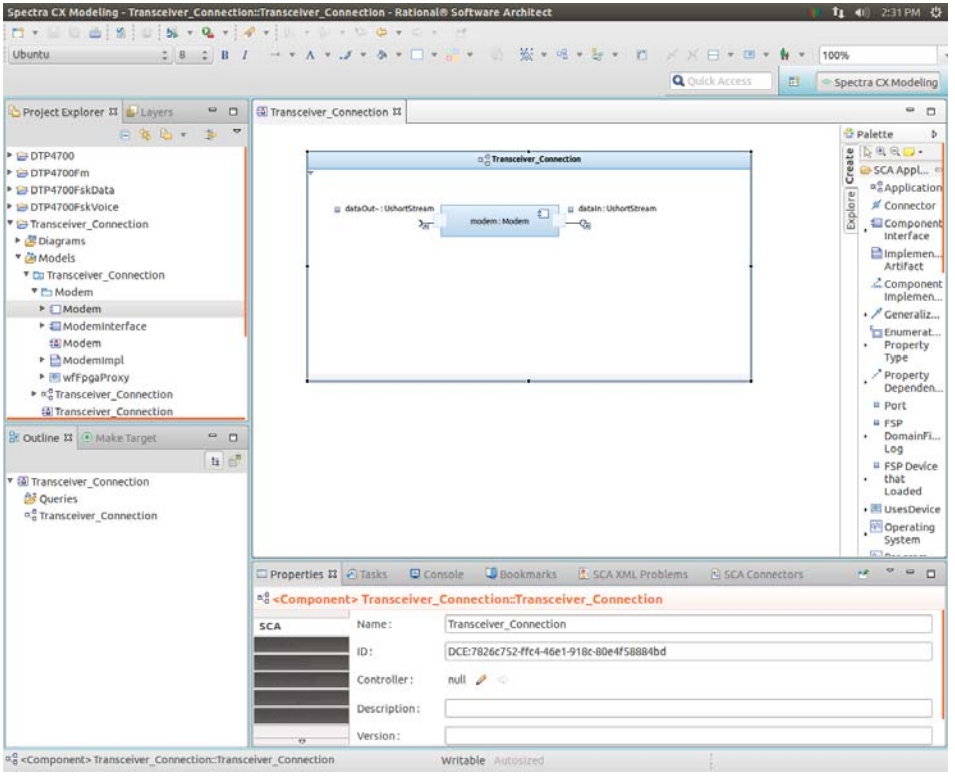

Figure 20 Defining an Application

Step 9: Open the diagram for the `Modem` component as shown in *Figure 19*. With the UsesDevice element under the SCA Application drawer of the Palette view selected, left-click on the Modem diagram to create an instance of a uses device relationship. Select the UsesDevice element in the diagram, and in the Properties view name the UsesDevice model element `wfFpgaProxy`.

Step 10: Ensure that the diagram for the `Modem` component is visible. With the UsesDevice Dependency element under the SCA Application drawer of the Palette view selected, draw a line connecting the structural realisation of the `Modem` to the wfFpgaProxy UsesDevice model element in the diagram. Select the UsesDevice Dependency element in the diagram, and in the Properties view name the UsesDevice Dependency model element `usesWfFpgaProxy`.

Step 11: Ensure that the diagram for the `Modem` component is visible. Expand the DTP4700 project in the Project Explorer view and navigate to the `WfFpgaProxy` component definition. Left-click on the UsesWfFpgaProxyType property model element and drag the icon onto the diagram canvas for the `Modem`. With the Property Dependency element under the SCA Application drawer of the Palette view selected, draw a line connecting the wfFpgaProxy UsesDevice to the UsesWfFpgaProxyType model element in the diagram. Select the Property Dependency element in the diagram, and in the Properties view enter the value '`1`'. This is shown in *Figure 21*.

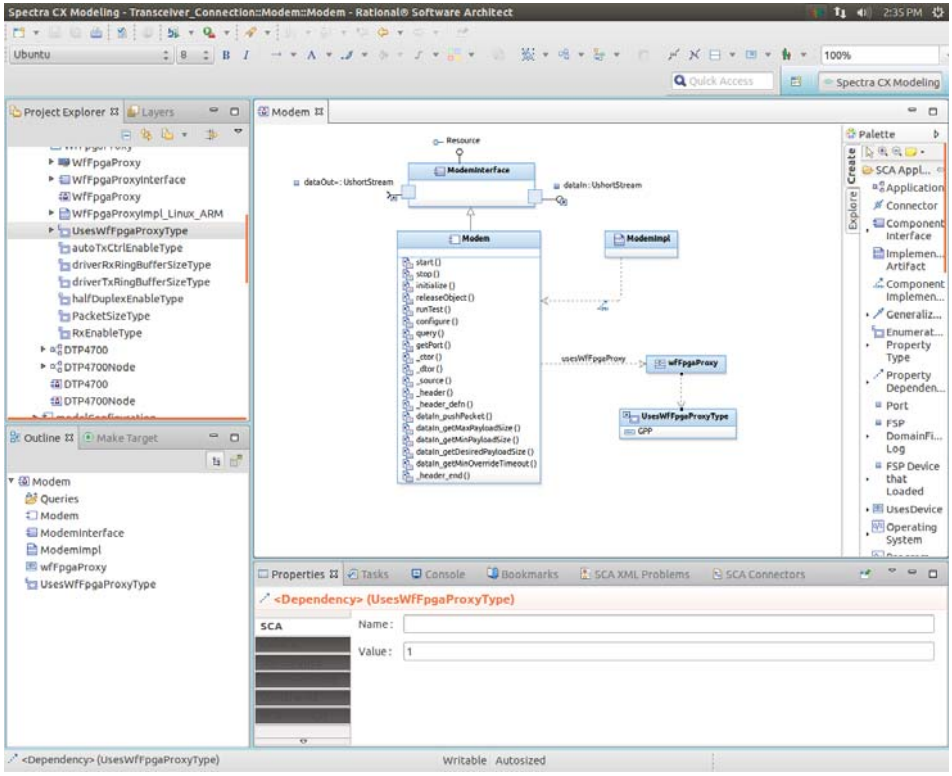

Figure 21 Uses Device Dependency

Step 12: Ensure that the Transceiver_Connection diagram containing the Transceiver_Connection application model element is visible. With the Free Standing Port (FSP) Device Used By element under the SCA Application drawer of the Palette view selected, left-click within the bounding box of the Transceiver_Connection Application to create an instance of a used by relationship for component ports included in the application.

Step 13: Select the Free Standing Port (FSP) Device Used By element in the diagram and then, in the Properties view, set the Free Standing Port (FSP) Device Used By element properties as follows:

Name: `wffpga_rx`

Findby Port Name: `wffpga_proxy_uses_port`. ***Note***: the port name MUST match (string equivalent) the required port on the `WfFpgaProxy` component.

Part: `modem – Transceiver_Connection::Transceiver_Connection::modem`

Uses Device: `usesWfFpgaProxy – Transceiver_Connection::Modem::usesWfFpgaProxy`

Conjugation: `Uses`

Porttype: `UshortStream – Packet::Packet::Packet:UshortStream`

Step 14: With the Connector element under the SCA Application drawer of the Palette view selected, link the `dataIn` port to the `wffpga_rx` FSP.

Step 15: Repeat steps 12 to 14 to model a `wfpga_tx` connection to the `WfFpgaProxy` component to send data to the RF/Transceiver sub-system from the `Modem`'s `dataOut` port.

The Free Standing Port (FSP) Device Used By element should have the following properties:

Findby Port Name: `wffpga_proxy_provides_port`

Part: `modem – Transceiver_Connection::Transceiver_Connection::modem`

Uses Device: `usesWfFpgaProxy – Transceiver_Connection::Modem::usesWfFpgaProxy`

Conjugation: `Provides`

Porttype: `UshortStream – Packet::Packet::Packet:UshortStream`

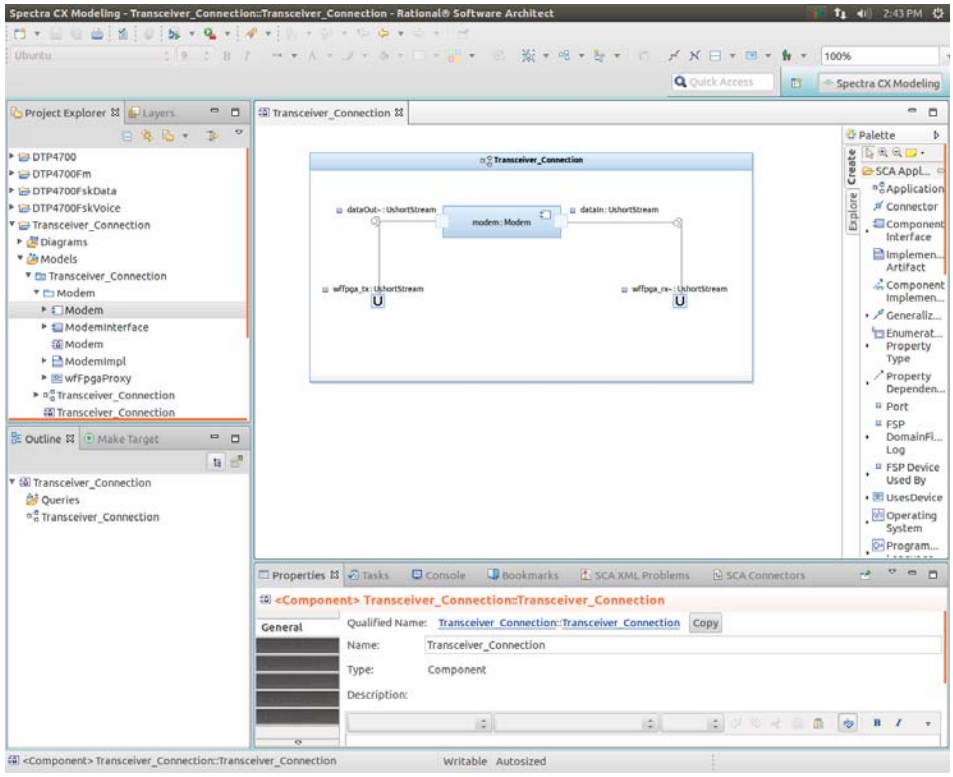

Figure 22 RF/Transceiver sub-system data link

CHAPTER

7 DTP4700 Probe Toolbox

The DTP4700 LiveUSB system contains a full installation of the Spectra DTP4700 Probe Toolbox including the ProbeViz visualization software which is installed into IBM Rational Software Architect alongside Spectra CX.

i Documentation for the Probe Toolbox software can be found in the PrismTech menu under Spectra > Thunder > SDR Probe Toolbox User Manual / SDR Probe Toolbox API Documentation.

This section of the *User Guide* describes additional components specifically provided to support the DTP4700 system.

7.1 DSP Probe Support

The Probes API for DSP requires a GPP-based probe listener to be running before test and latency probes can be registered. The DTP4700 SCA platform includes the `DSPProbeListenerService` device which starts the required listeners during platform deployment and stops the listeners when the platform is shut down. This allows users of the DTP4700 system to add test and latency probes to a DSP-based component without having to perform any extra initialization on the GPP.

7.2 Spectra CX Build Configurations

Additional Spectra CX build configurations are provided for both the ARM and DSP processors. These build configurations can be used to generate a build environment pre-configured with the necessary compiler settings to support software probes. *Table 7* lists the build configurations which include probe support.

Table 7 Spectra CX build configurations including probe support

Build Configuration	Target
`eorb-dspbios-cl6x::eorb20-dspbios-cl6x-linux-c-dtp4700+probes-build`	DSPBIOS/DSP
`eorb20-linux-gcc-arm::eorb20-linux-gcc-arm-dtp4700+msgqiop+probes-build`	TILinux/ARM

Debug versions of the build configurations listed in *Table 7* are also provided.

The build configurations listed in *Table 7* set a compiler define named `DTP_PROBES`. Through the use of preprocessor directives this compiler define can be used to optionally add software probes into a component. The FM Waveform example uses this compiler define in this way so that probes are only registered by the components if one of the build configurations in *Table 7* is used.

7.3 ProbeViz

The ProbeViz visualization tool included with the DTP4700 LiveUSB system can be accessed from within IBM Rational Software Architect. To access ProbeViz, open the ProbeViz perspective within RSA.

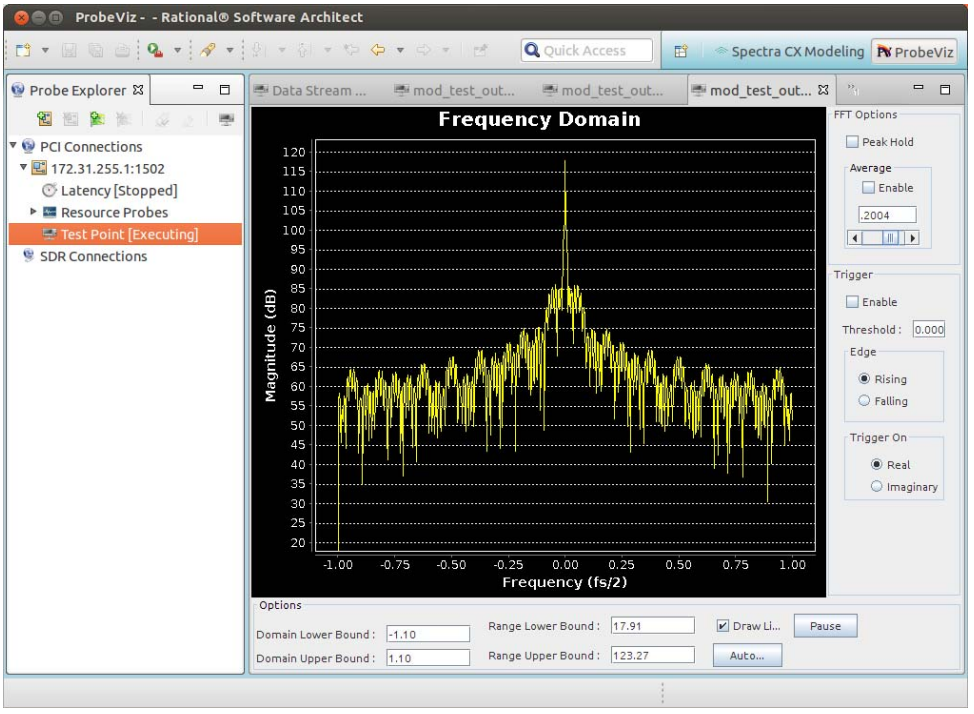

Figure 23 ProbeViz visualization tool

7.4 Example Waveform

The example FM Waveform included with the DTP4700 system is instrumented with test and latency probes. Data from these probes can be accessed *via* ProbeViz when the waveform is running.

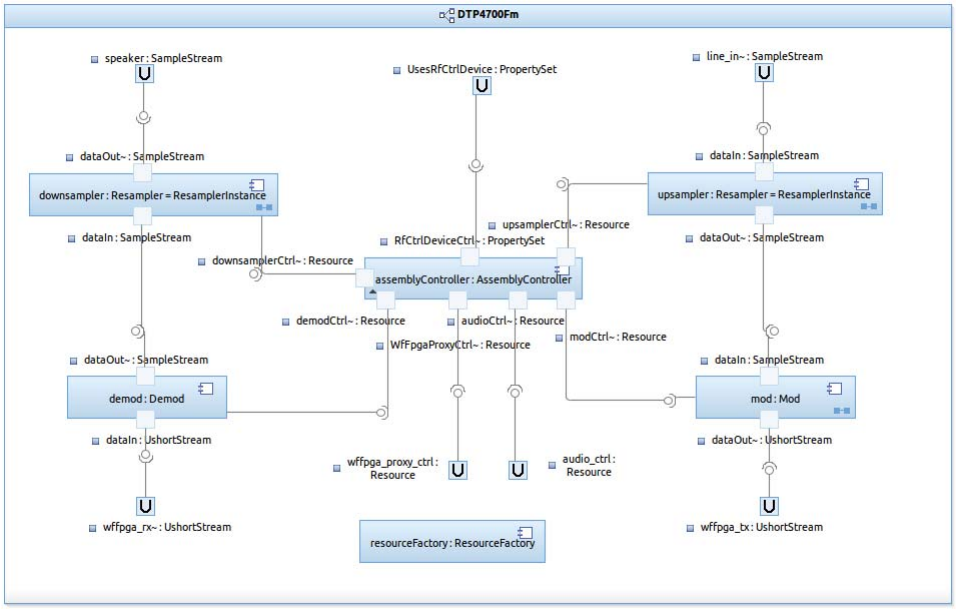

Figure 24 DTP4700Fm Application Assembly model

Table 8 below lists the test probes and *Table 9* lists the latency probes that are registered by the FM waveform example. All the test probes registered by the FM waveform are *extraction* probes.

Table 8 Test probes registered by the FM waveform

Name	Data Type	Data Source
upsampler_test_in	Signed 16 bit Integer	GPP
upsampler_test_out	Signed 16 bit Integer	GPP
mod_test_in	Signed 16 bit Integer	DSP
mod_test_out	Complex Signed 16 bit Integer (I & Q)	DSP
demod_test_in	Complex Signed 16 bit Integer (I & Q)	DSP
demod_test_out	Signed 16 bit Integer	DSP
downsampler_test_in	Signed 16 bit Integer	GPP
downsampler_test_out	Signed 16 bit Integer	GPP

Table 9 Latency probes registered by the FM waveform

Group Number	Probe Number	Location
1 (Tx)	1	upsampler dataIn
	2	upsampler dataOut
	3	mod dataIn
	4	mod dataOut
2 (Rx)	1	demod dataIn
	2	demod dataOut
	3	downsampler dataIn
	4	downsampler dataOut

7.4.1 Probe Registration

7.4.1.1 GPP Probe Registration

The probes API on the GPP allows for individual probes to be registered independently, the only restriction being that each probe name (in the case of test probes) or number (in the case of latency probes) is unique.

In the example FM waveform the `upsampler` and `downsampler` component instances register probes. The `upsampler` and `downsampler` are instances of the `Resampler` component configured differently to carry out upsampling/downsampling respectively. Probes are registered in the `Resampler` component's constructor based on the name of the component instance. Probes are unregistered in the `Resampler` component's destructor.

7.4.1.2 DSP Probe Registration

The probes API on the DSP requires that for each type of probe all instances are registered at the same time. This means that while test and latency probe types can be registered separately, all test probes must be registered at the same time and all latency probes must also be registered at the same time.

The example FM waveform deploys two components, `Mod` and `Demod`, to the DSP. As the DSP can only run a single binary at any one time these components are deployed *via* the ResourceFactory component. As probes must be registered at the same time, probe registration occurs in the `init` function of the `ResourceFactory` component. Probes are unregistered in the `fini` function of the `ResourceFactory` component.

7.4.2 Latency Probes

The example FM waveform contains two groups of latency probes. Group 1 monitors latency through the Tx pipeline and group 2 monitors latency through the Rx pipeline. Latency probes within a group must be called in sequence. As each part of the Tx and Rx pipelines can be active simultaneously latency is measured every 50 packets to ensure that each probe is called in sequence.

CHAPTER

8 SCA Waveform Tutorial

This chapter uses the supplied FM waveform to illustrate the general procedures required to:

- *generate, build, and compile component assemblies using a Spectra CX model,*
- *prepare a domain profile on the target filesystem (SD card) in readiness for running an SCA application on DTP4700*
- *connect the Spectra CX Monitor running on the host to the DTP4700 SCA platform on the target to control the life cycle of waveform applications*

The DTP4700 is delivered with an SD card already prepared, including the demo waveforms. To run the supplied FM waveform example directly and bypass Spectra CX modelling and code generation steps, please see Section 8.2.10, Running the FM Waveform, on page 88.

8.1 The FM Waveform Model

The Spectra CX model for the `DTP4700Fm` application assembly, as described in Section *5.2* on page 41, will be used as the basis for this tutorial. The components used in the application assembly realize ports and properties designed to interconnect components to form the audio data path and to control the operation of the audio waveform.

Table 10 Component Port Properties

Component	Ports	Properties
`Mod`	***`dataIn`***: Exposes the *`SampleStream`* interface for incoming *`UshortStream`* data and control	***`gain`***: A CORBA::short to hold the modulator gain. The gain value is multiplied to each modulator complex I&Q output pair
	`dataOut`: Sends interleaved I&Q data *via* the *`UshortStream`* interface	

Table 10 Component Port Properties

Component	Ports	Properties
Demod	***dataIn***: Exposes the *UshortStream* interface for incoming interleaved I&Q data and control	No configurable properties
	dataOut: Sends sample data *via* the *SampleStream* interface	
Resampler	***dataIn***: Exposes the *SampleStream* interface for incoming *UshortStream* data and control	***numerator***: A *CORBA::short* to hold the FIR filter numerator value. The numerator value is used in rate conversion such that output rate = input rate (numerator/denominator)
	dataOut: Sends filtered data *via* the *SampleStream* interface	***denominator***: A *CORBA::short* to hold the FIR filter denominator value. The denominator value is used in rate conversion such that output rate = input rate (numerator/denominator)
		filterGainAdj: A *CORBA::short* to hold the gain adjust for the FIR filter [1]
		FilterCoeffs: A sequence of *CORBA::short* values to hold the fixed-point filter coefficients for the FIR filter

1. The `filterGainAdj` property on the `Resampler` is set with a default gain at the output of the filter of `-32768` to adjust the dynamic range. The gain adjustment is applied in addition to the default gain. A `filterGainAdj` of `0` would leave the default gain unchanged.
Each filter output sample is shifted by N bits, where N > 0, the shift is to the left; where N < 0, the shift is to the right, and N is given by:
`filterGainAdj` > 0: round(log base 2(`filterGainAdj`)) – 15
or
`filterGainAdj` < 0: round(-log base 2(`-filterGainAdj`)) – 15

8.2 The FM Tutorial Step-by-step

8.2.1 Starting Spectra CX

For convenience a link is provided under the PrismTech menu under **Spectra > DTP4700 > Workspaces > Spectra CX (DTP4700 workspace)** on the host development system desktop that can be used to open a pre-existing Spectra CX workspace containing all the supplied model projects.

If starting Spectra CX using the above link, please continue this tutorial from Section 8.2.4, *Generating DTP4700 Platform Artefacts*, on page 73.

Alternatively to run Spectra CX creating a new workspace, choose **IBM Software Delivery Platform > IBM Rational Software Architect 9.1** from the PrismTech menu, then continue this tutorial from Section 8.2.2, *Choosing a Workspace*, below.

8.2.2 Choosing a Workspace

When you start Spectra CX a dialog appears for you to choose where to locate your workspace. A workspace is a folder where the projects that you create will be stored. If you do not specify a workspace, a default workspace will be created for you.

Click **OK** when you have decided on your workspace location.

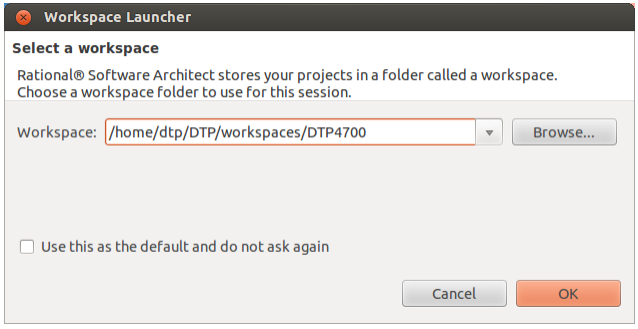

Figure 25 Select or create a workspace

The first time you open Spectra CX or create a new workspace, you will see a 'Welcome' screen.

Close the Welcome screen by clicking **X** on the tab or by clicking the Workbench icon (circled in *Figure 26* below).

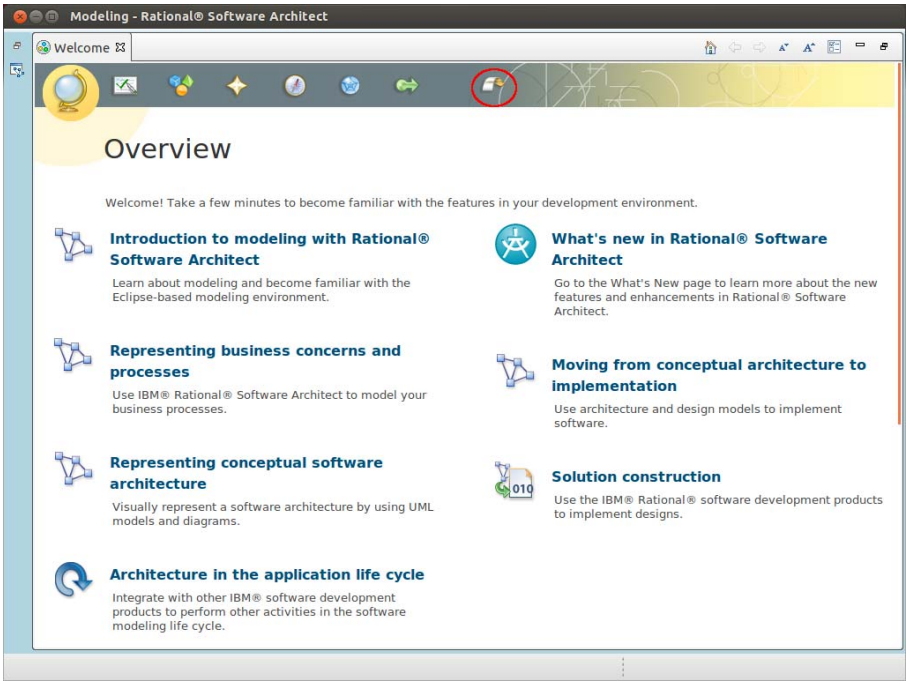

Figure 26 The 'Welcome' screen

The illustration overleaf (*Figure 27*) shows the default Workbench layout before any projects have been created.

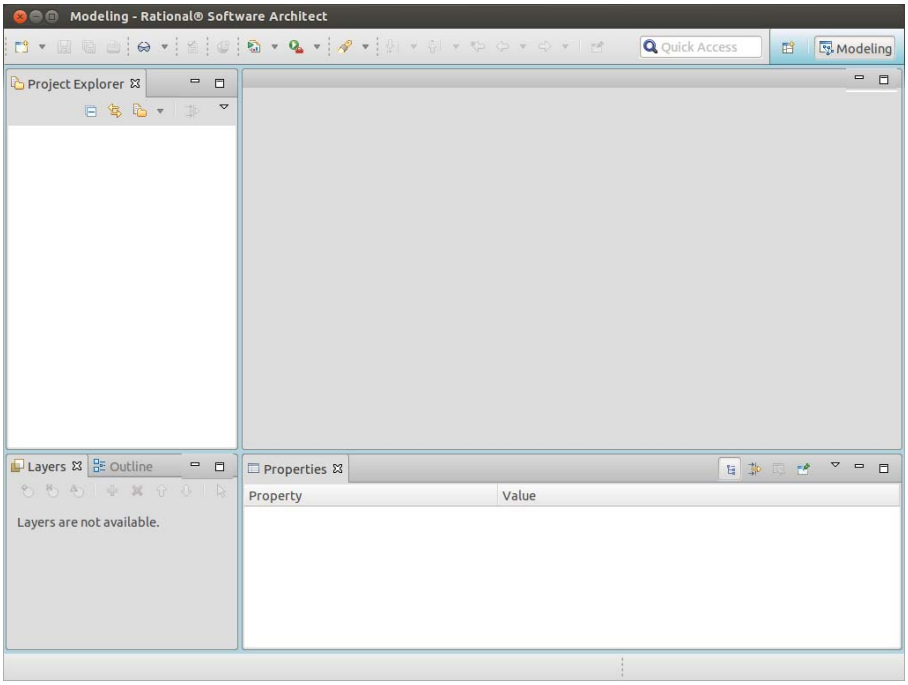

Figure 27 Default Workbench showing empty workspace

If you already have projects under development in Spectra CX then those projects are displayed in the Project Explorer pane. If, however, this is your first project, then the Project Explorer will show no projects in your workspace.

To work with DTP4700 models, the tool must be switched to the Spectra CX Modeling perspective.

Figure 28, *Figure 29* and *Figure 30* show how to switch into the Spectra CX Modeling perspective.

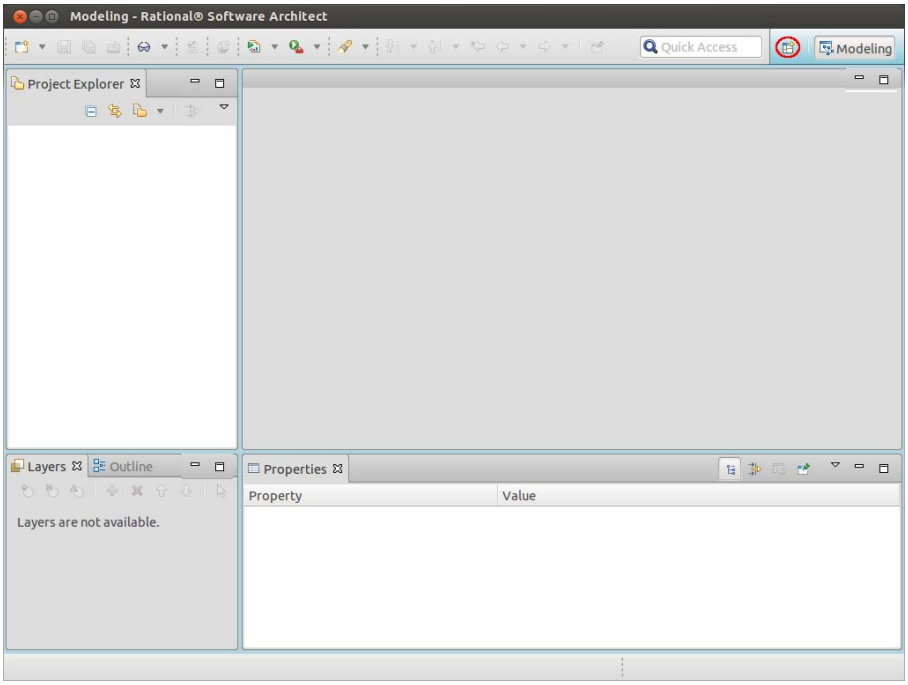

Figure 28 Click the Open Perspective button (circled)

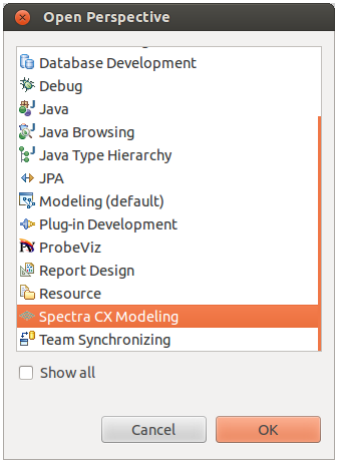

Figure 29 Open the Spectra CX Modeling perspective

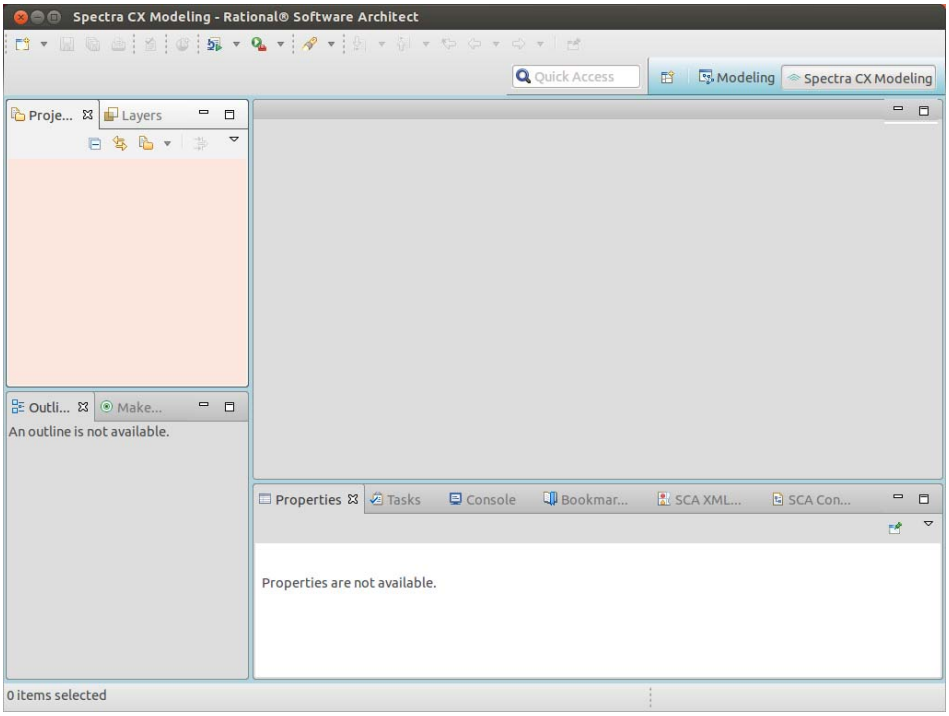

Figure 30 The Spectra CX Modeling environment ready to start

8.2.3 Importing the DTP4700 Tutorial Packages

To import the DTP4700 tutorial packages into the workspace, perform the following steps.

Step 1: On the menu bar, choose **File > Import**; the dialog shown in *Figure 31* will appear.

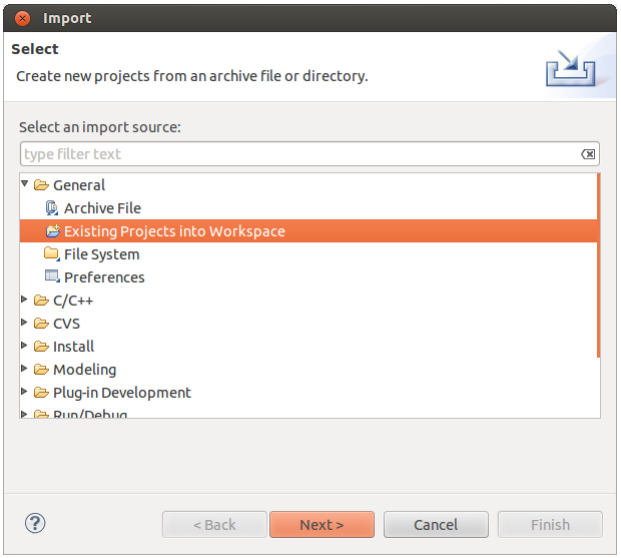

Figure 31 The Import Wizard

Step 2: Select Existing Projects into Workspace and click **Next**.

Step 3: Click the radio button for Select archive file and click the **Browse** button.

Step 4: Locate and select the `zip` file that contains the DTP4700 projects (projects are stored in `/opt/dtp/models.zip`).

i The `models.zip` archive contains five model projects:

- *`DTP4700`*: the DTP4700 SCA platform model.
- *`DTP4700Fm`*: the analog FM audio waveform used in this tutorial.
- *`DTP4700FskData`*: the data waveform which uses FSK modulation.
- *`DTP4700FskVoice`*: the digital audio waveform which uses CVSD encoding and FSK modulation.
- *`x86`*: additional platform nodes that can be used to deploy application components to the host development system.

Step 5: Click **OK** to use the selected archive.

Step 6: Click **Finish** to complete the import operation.

Step 7: The DTP4700 Platform and Tutorial FM waveform now appear in the Project Explorer pane.

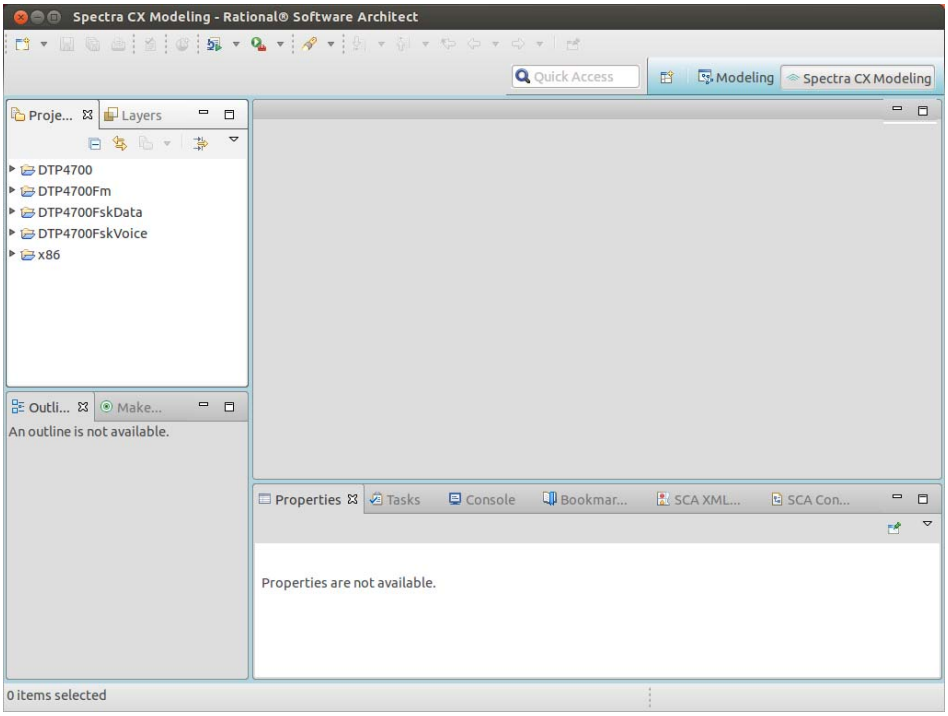

Figure 32 Project Explorer showing imported projects

8.2.4 Generating DTP4700 Platform Artefacts

This section describes how to generate the domain profile and code for the DTP4700 platform.

Step 1: In the Project Explorer, expand the DTP4700 project, navigate to **Models > DTP4700 > DTP4700**, and double-click the DTP4700 diagram element in the tree.

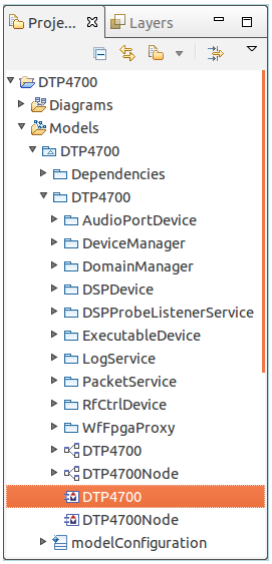

Figure 33 DTP4700 diagram element selected in Project Explorer

This will display the diagram showing the platform model element containing an instance of the `DTP4700Node` named `dtp4700Node`. This is shown in *Figure 34*.

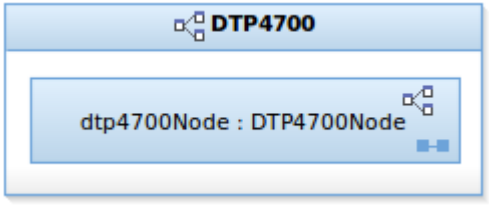

Figure 34 The DTP4700 Platform Model Element

Step 2: Double-click the DTP4700Node diagram element in the tree. This will display the diagram showing the `DTP4700Node` and the component instances contained within it, as shown in *Figure 35*.

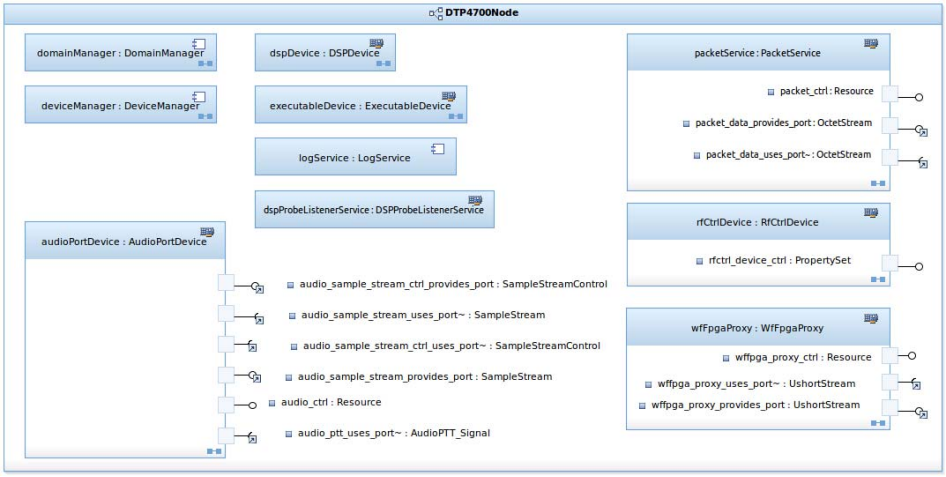

Figure 35 Components in the DTP4700Node

8.2.4.1 Generating DTP4700 SCA Descriptors

This section describes how to generate the XML domain profile for the DTP4700 platform.

Step 1: In the DTP4700 diagram, right-click on DTP4700 platform to open its context menu, then choose **Generate > Descriptor(s)**.

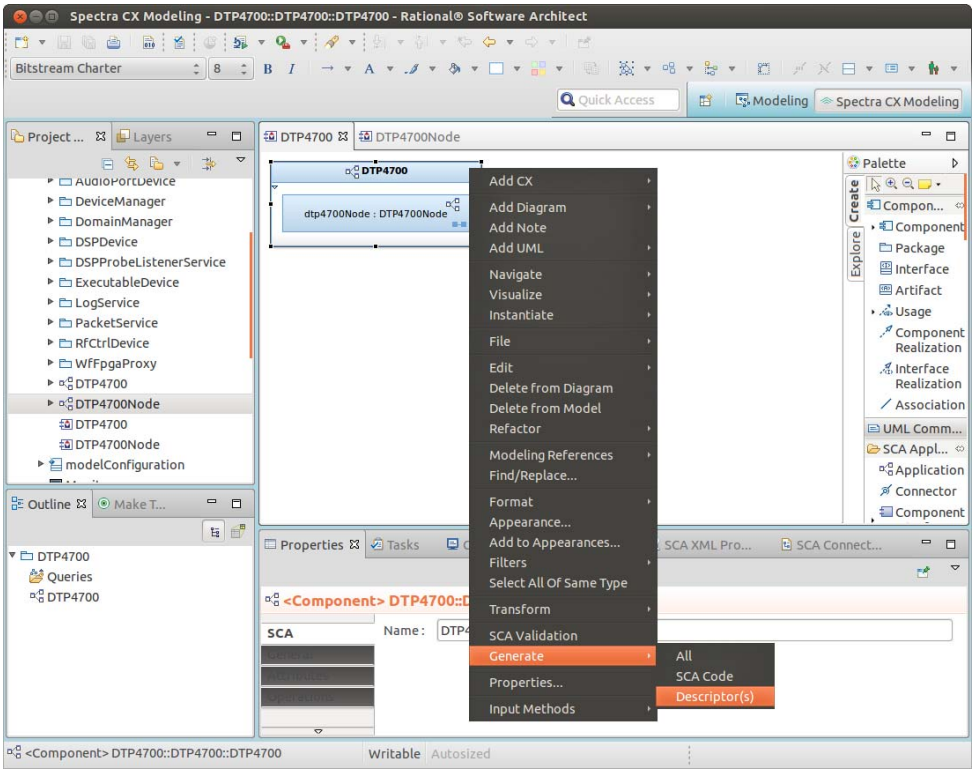

Figure 36 Generating SCA Descriptors

When the platform descriptors have been generated, the console will show validation status, errors, and warnings.

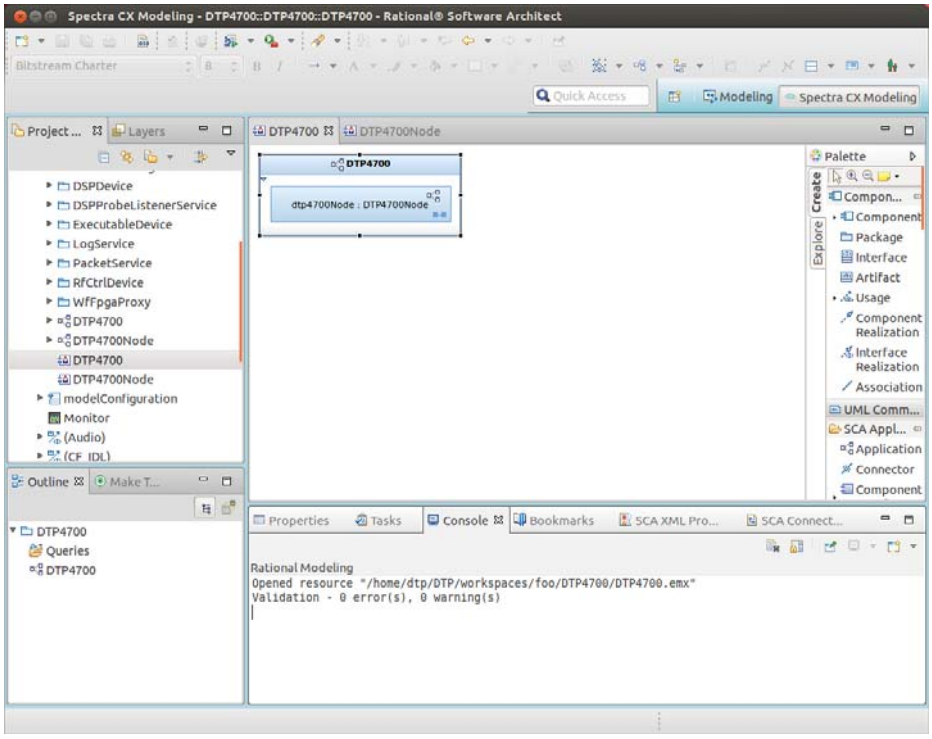

Figure 37 After generating Platform Descriptors

In *Figure 37* the Console tab shows the results of the Model Validation Procedure.

8.2.4.2 Generating DTP4700 SCA Source Code

This section describes how to generate source code for the DTP4700 platform.

Step 1: On the DTP4700 diagram, right-click the DTP4700 platform component, then choose **Generate > SCA Code** from the context menu.

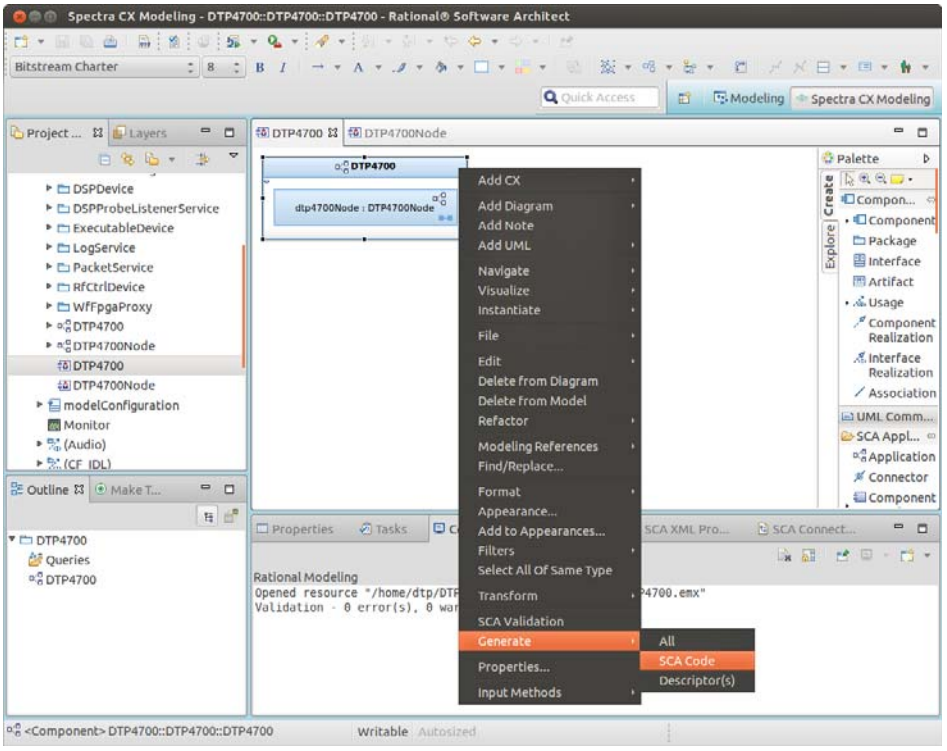

Figure 38 Generating SCA Source Code

Following the generation of descriptors and binaries from the platform model element, the Project Explorer view in Spectra CX will display two new folders in the workspace containing the generated files. The convention for folder names is configurable using Spectra CX, in this case `DTP4700_src` will contain the descriptors, and `DTP4700_src_CPP` will contain the binaries. Note in *Figure 39* that the code generation procedure also generated the required build instructions resulting in the new elements being displayed in the Make Target view.

i Please note that the DTP4700 platform models reference existing binaries and therefore no source code will be generated. During generation the pre-existing binaries are copied into the `obj` directory underneath the source project. If the platform is extended then source code will be generated for any new components.

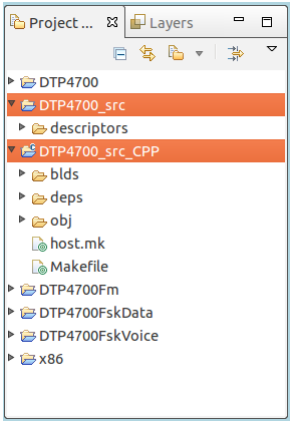

Figure 39 Workspace projects containing the generated files

8.2.5 Building the DTP4700 Platform

To build the DTP4700 platform:

Step 1: Left-click the DTP4700_src_CPP folder in the Make Target view to expand the target elements.

Step 2: Double-click the **all** target (see *Figure 40*).

i Please note that as no code is generated for the DTP4700 platform components then during the Make Targets step no code will be compiled. Instead the makefile copies the pre-existing binaries referenced by the components into the `obj` directory underneath the source project.

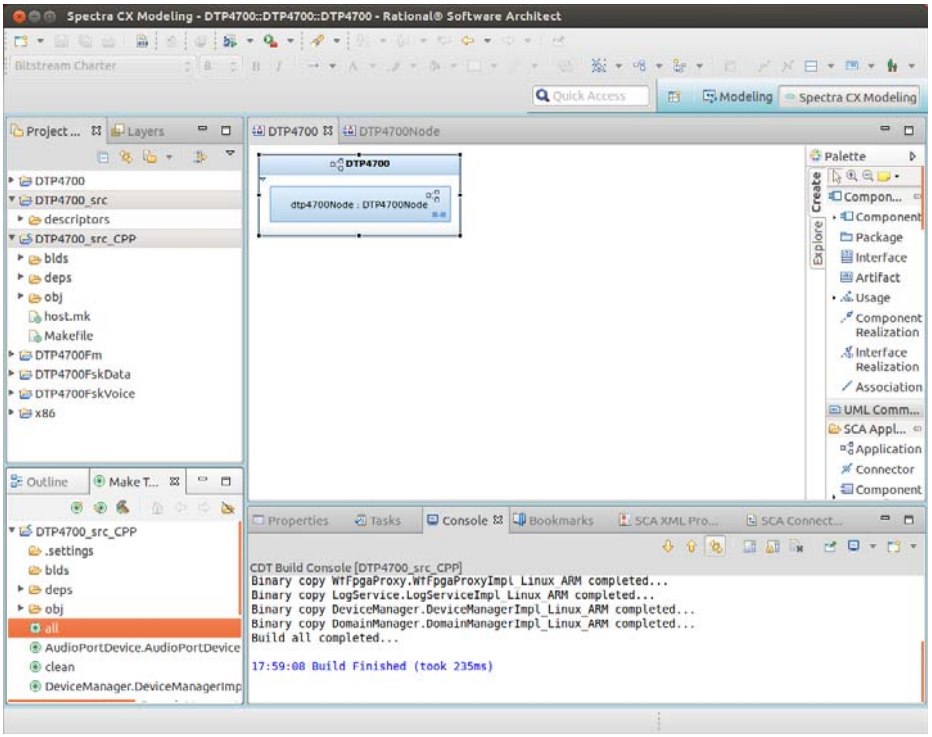

Figure 40 Building the DTP4700 platform

8.2.6 Generating the FM Waveform Artefacts

This section describes how to generate profile descriptors and source code for the tutorial FM waveform.

Step 1: In the Project Explorer, expand the folders DTP4700Fm > Models > DTP4700Fm > DTP4700Fm, and double-click the DTP4700Fm diagram model element to display the application assembly diagram as shown in *Figure 41*.

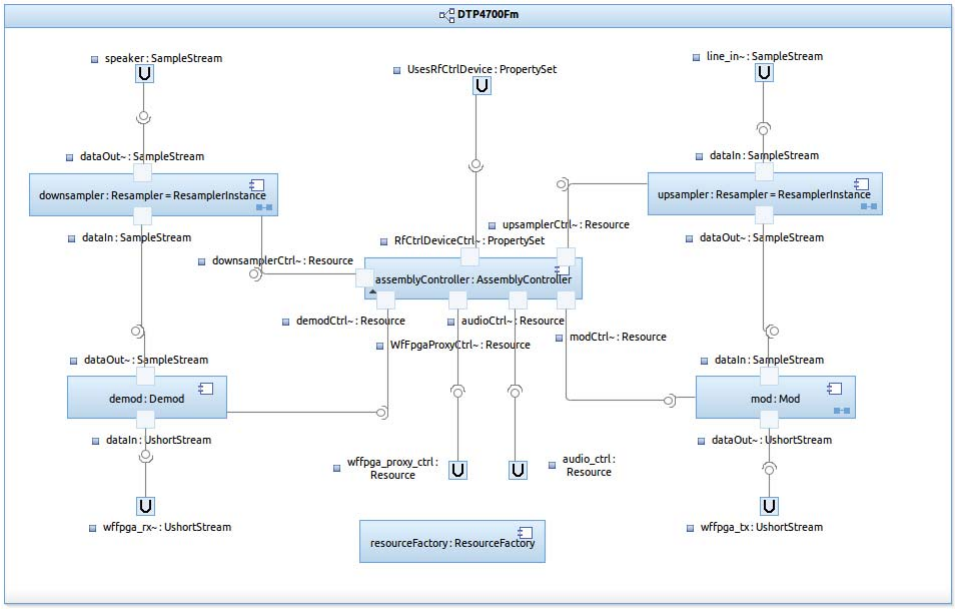

Figure 41 Application Assembly Diagram for the FM waveform

8.2.6.1 Generating FM Waveform SCA Descriptors

To generate domain profile descriptors for the FM tutorial waveform, right-click the DTP4700Fm application and then choose **Generate > Descriptor(s)** from the context menu.

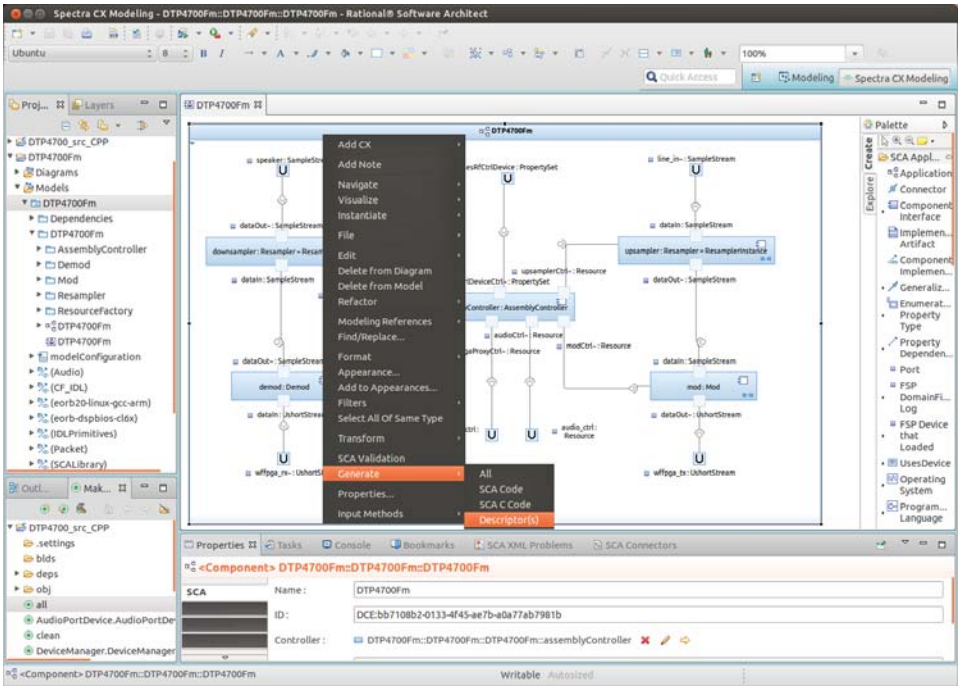

Figure 42 Generating Descriptors

8.2.6.2 Generating FM Waveform SCA Source Code

The DTP4700Fm project contains a combination of C and C++ components. To generate source code for the tutorial FM waveform:

Step 1: To generate C++ code, right-click the DTP4700Fm application and choose **Generate > SCA Code** from the context menu.

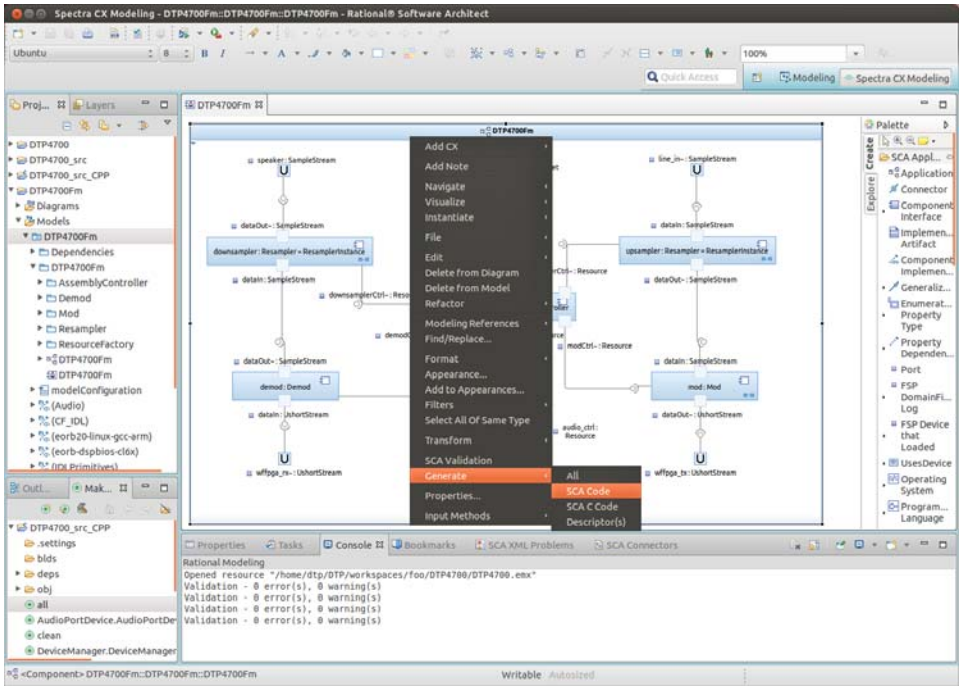

Figure 43 Generating C++ source code

Step 2: To generate C code, right-click the DTP4700Fm application and choose **Generate > SCA C Code** from the context menu.

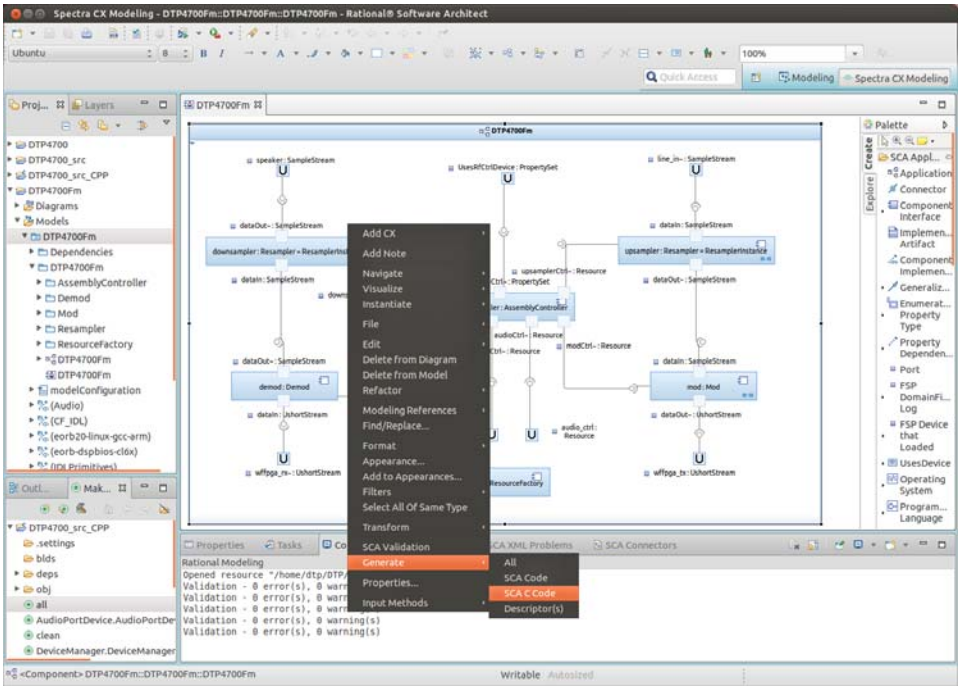

Figure 44 Generating C source code

i Both C and C++ code, as well as descriptors, can also be generated by choosing **Generate > All**.

When the waveform descriptors and SCA code have been generated, projects containing the generated files will appear in the Project Explorer view:

DTP4700Fm_src contains the descriptor files

DTP4700Fm_src_CPP contains the C++ code files

DTP4700Fm_src_C contains the C code files

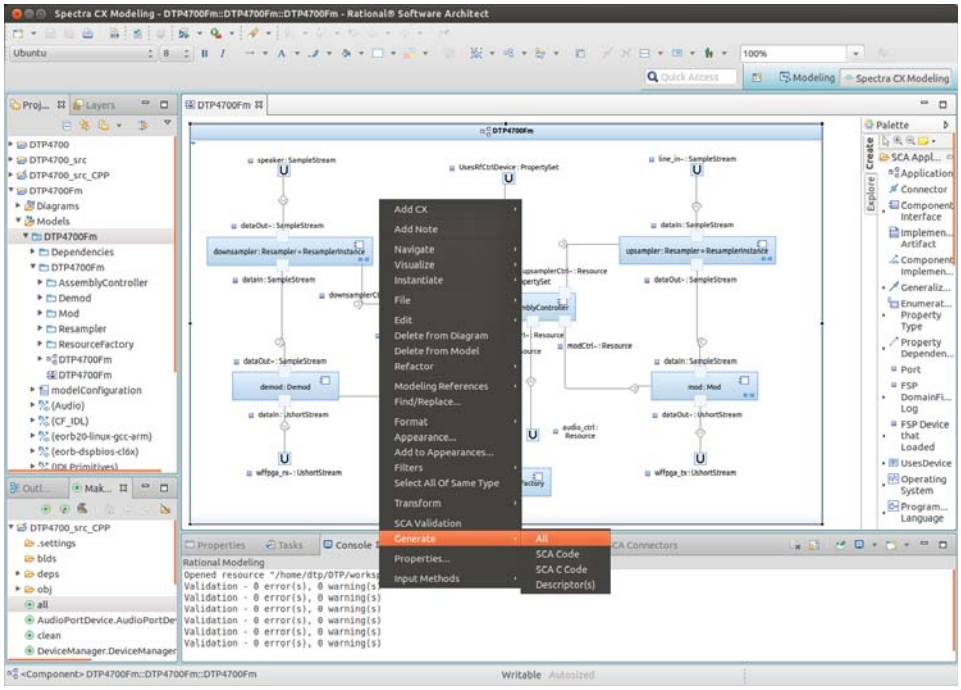

Figure 45 Generating All

Figure 46 shows the generated projects in the Project Explorer.

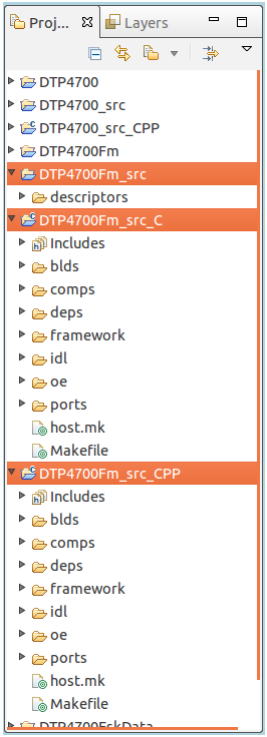

Figure 46 Generated projects

8.2.7 Building the FM Waveform

To build the waveform:

Step 1: Click to expand the DTP4700Fm_src_CPP folder in the Make Target view.

Step 2: Double-click on the all target element to compile and link the C++ source code.

Step 3: Click to expand the DTP4700Fm_src_C folder in the Make Target view.

Step 4: Double-click on the all target element to compile and link the C source code.

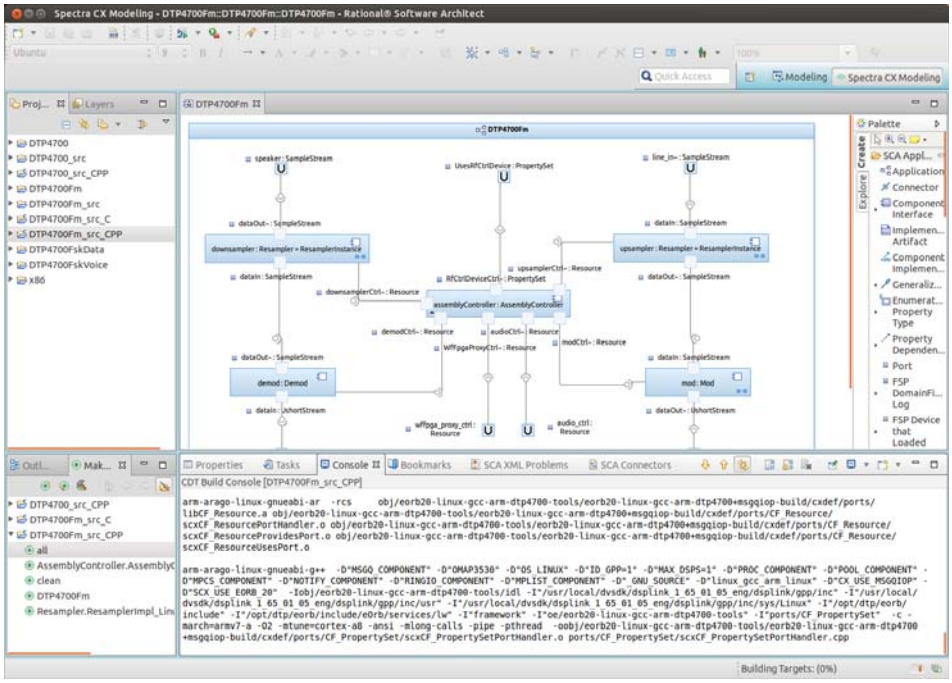

Figure 47 Compiling and Linking

8.2.8 Packaging XML and Binaries for Deployment

Before running the generated platform and waveform it is first necessary to package the generated XML and binaries into a bundle on the target file system which is suitable for deployment.

8.2.8.1 Platform

Step 1: Open the Ubuntu File Browser by clicking the **Home Folder** icon on the Unity Launcher.

Step 2: From the Computer list in the File Browser select File System.

Step 3: Navigate to the target file system: mnt > targetfs.

Step 4: Navigate to the opt > dtp > platform directory on the target file system.

Step 5: In Spectra CX select the platform `descriptors` folder (found under the DTP4700_src project) and drag the folder into the `platform` directory in the Ubuntu file browser.

Step 6: In the Ubuntu File Browser rename the `descriptors` folder to be `xml-DTP4700-tutorial`.

i The generated XML is configured to expect the platform binaries to be in the `../bin` directory. As the platform binaries are already present in the appropriate location there is no need to copy the binaries onto the target file system.

8.2.8.2 Waveform

Step 1: Open the Ubuntu File Browser by clicking the **Home Folder** icon on the Unity Launcher.

Step 2: From the Computer list in the File Browser select File System.

Step 3: Navigate to the target file system: mnt > targetfs.

Step 4: Navigate to the opt > dtp > applications directory on the target file system.

Step 5: In Spectra CX select the waveform `descriptors` folder (found under the DTP4700Fm_src project) and drag the folder into the `applications` directory in the Ubuntu file browser.

Step 6: In the Ubuntu File Browser rename the `descriptors` folder to be `DTP4700Fm-tutorial`.

Step 7: Copy the `Demod.out`, `Mod.out` and `ResourceFactory.out` binaries from Spectra CX (found under the `obj` directory in the DTP4700Fm_src_C project) to the `DTP4700Fm-tutorial` directory in the Ubuntu File Browser.

Step 8: Copy the `AssemblyController` and `Resampler` binaries from Spectra CX (found under the `obj` directory in the DTP4700Fm_src_CPP project) to the `DTP4700Fm-tutorial` directory on the Ubuntu File Browser.

8.2.9 Creating a DTP4700 SD Card

A utility script (`mksdboot.sh`) is provided in the `/opt/dtp/bin` directory of the LiveUSB system to create a bootable SD card for the DTP4700 system. Please refer to Section 3.1.1.5, *Utility Scripts*, on page 16, for instructions on using this script.

The script will create a file system consisting of the TI Linux operating system and a copy of the target file system from the host. The target file system contains several platform and waveforms including the ones copied onto the target file system in Section 8.2.8, *Packaging XML and Binaries for Deployment*, on page 87.

i When running the `mksdboot.sh` script you will be asked to choose which platform version you would like to be run when the target system boots. Select 3: Other and then enter `xml-DTP4700-tutorial` as the name of the platform.

Ensure the SD card is plugged into the Thunder system's SD/MMC slot before booting the target system.

8.2.10 Running the FM Waveform

This section describes how to run the compiled example FM Waveform on the DTP4700 platform.

8.2.10.1 Configuring Ubuntu Host Network

By default the Ubuntu host system is configured to acquire an IP address *via* DHCP, but the Thunder system is configured with a static IP address. In such a configuration the two systems will not be able to communicate.

To configure the Ubuntu host system with a static IP address compatible with the Thunder target:

Step 1: Left-click the PrismTech icon on the system menu to activate the PrismTech menu.

Step 2: From the PrismTech menu choose **Spectra > IP Address > Switch to Static IP**.

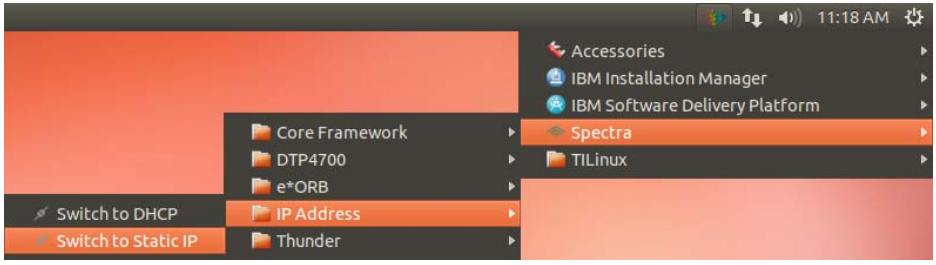

Figure 48 Setting a static IP address

8.2.10.2 Connecting Spectra CX Monitor to DTP4700

The monitor object in the model contains details of the DTP4700 and how to connect to it. The details of how to connect to a NameService and the name under which the SCA DomainManager is registered is contained in the Monitor. The values can be modified if required.

To connect to the platform, first open the SCA Monitor:

Step 1: In the Project Explorer, expand the folders DTP4700 > Models > DTP4700, and right-click on Monitor.

Step 2: Choose **Open SCA Monitor** (see *Figure 49*).

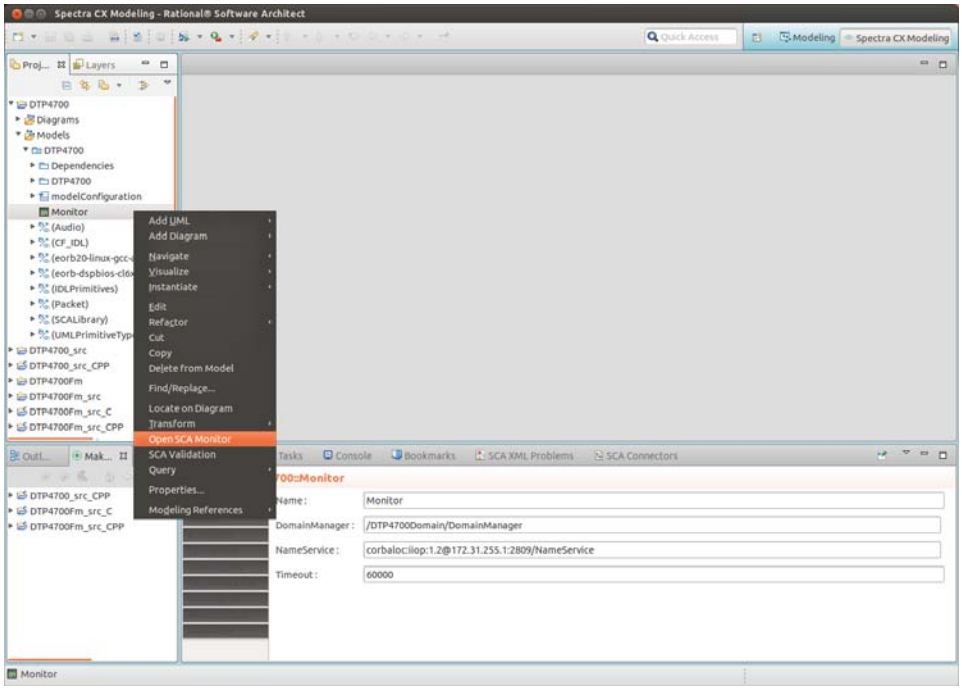

Figure 49 Opening the SCA Monitor

Connecting to the DTP4700 will display the running SCA platform, as shown in *Figure 50*.

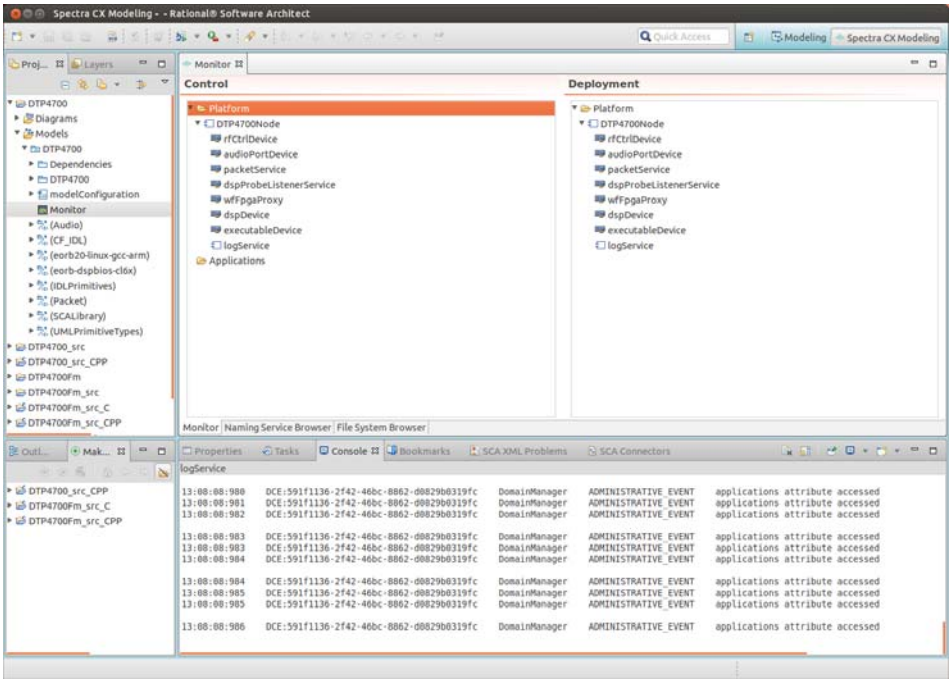

Figure 50 Monitor showing running SCA platform

i In the Monitor view, the `DomainManager` and `DeviceManager` are implicit, and are therefore not displayed in the Control or Deployment panes.

8.2.10.3 Installing the FM Waveform

To install the FM Waveform:

Step 1: Right-click **Applications** and from the pop-up menu choose **Install Application Factory**.

Step 2: Select /DTP4700Domain/DTP4700Node/applications/DTP4700Fm-tutorial/DTP4700Fm.sad.xml

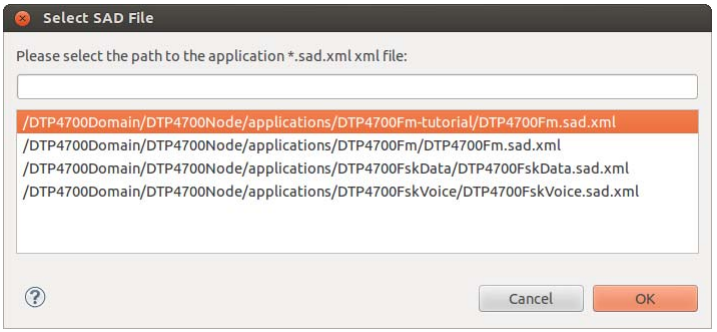

Figure 51 Installing the FM Waveform

Step 3: Click **OK**.

When the application is successfully installed, its Application Factory is displayed in the Control pane (see *Figure 52*).

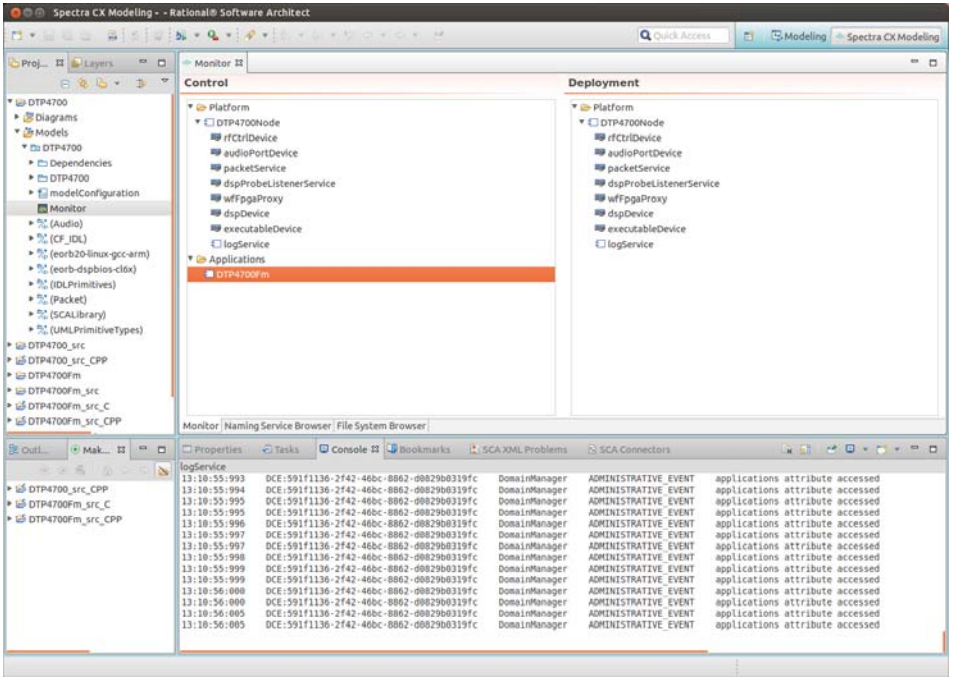

Figure 52 FM Waveform successfully installed

8.2.10.4 Creating the FM Waveform

To create an instance of the FM Waveform from the installed Application Factory:

Step 1: Right-click the DTP4700Fm application and choose **Create Application**.

Step 2: Enter a name for the instance (for this example, the name is "**a1**").

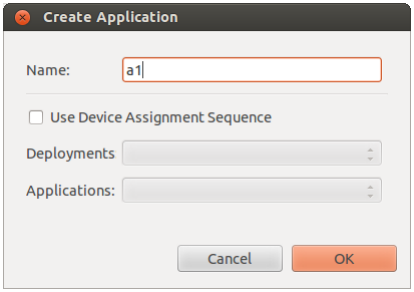

Figure 53 Naming the new application instance

Step 3: Click **OK**.

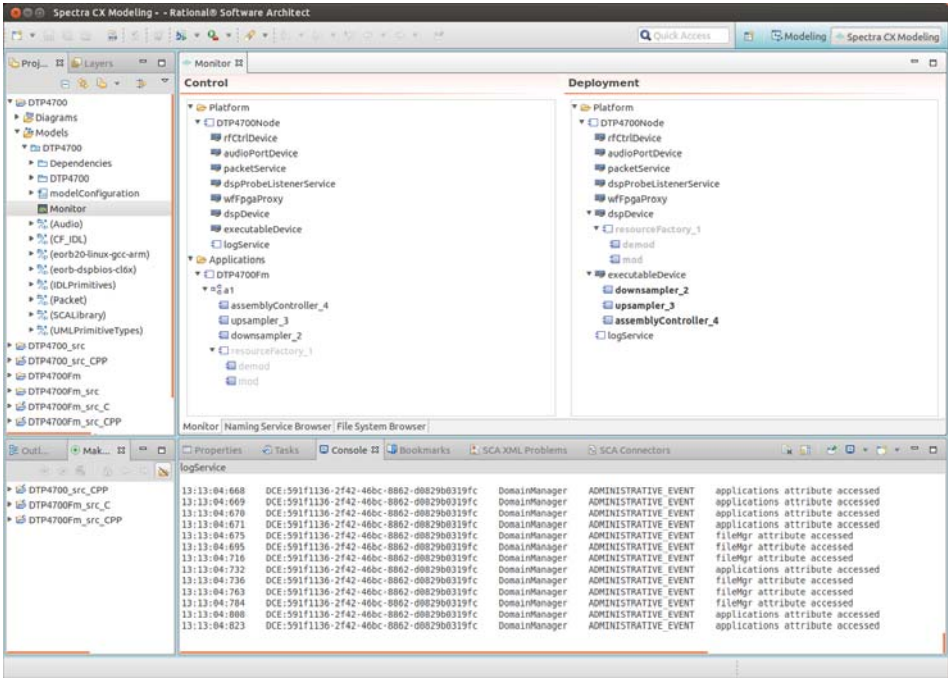

Figure 54 Successful creation of an application instance

Figure 54 shows a successful instantiation of the DTP4700Fm application. The Deployment pane shows the instantiated waveform components of the application a1 running on the DTP4700 SCA platform components.

i The components that were deployed on the DSP device appear greyed-out in the Monitor as these components do not implement the IIOP transport profile so the Monitor cannot communicate with them. In such a case the Monitor displays only the information present in the XML descriptors for the component. The `cfadmin` utility that runs on the Thunder target hardware can be used to view the details of all running waveform components including those based on transports other than IIOP. Please refer to Section 8.3, *Running the FM Tutorial using cfadmin*, on page 102 for more details.

8.2.10.5 Starting the FM Waveform

To start the newly-created instance of the application:

Step 1: Set the Transmit Frequency.

Select the rfCtrlDevice in the Control pane and switch to the Properties view, then select the SCA Properties tab. Select the txFrequency property and set the value to the frequency you wish to transmit on.

⚠ **NOTE:** If transmitting over the air, please ensure that the frequency chosen is legal in your country.

Step 2: Set the Receive Frequency.

Select the rfCtrlDevice in the Control pane and switch to the Properties view, then select the SCA Properties tab. Select the rxFrequency property and set the value to the frequency you wish to transmit on.

i To run the application in loopback mode the transmit and receive frequencies must be compatible. This could be achieved by setting the Rx and Tx frequencies to the same value. However it is recommended that the Rx frequency should be configured to be +60Hz or -60Hz of the Tx frequency. Please refer to the DTP4700 *Release Notes* for more details.

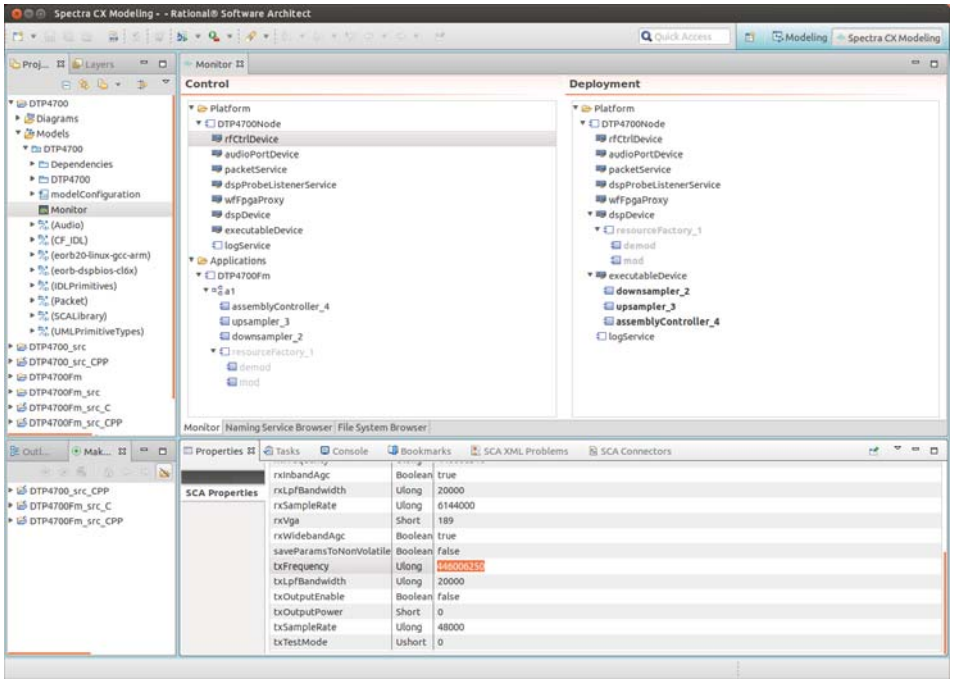

Figure 55 Setting the Transmit Frequency

Step 3: Choose **Start** from the application's context menu.

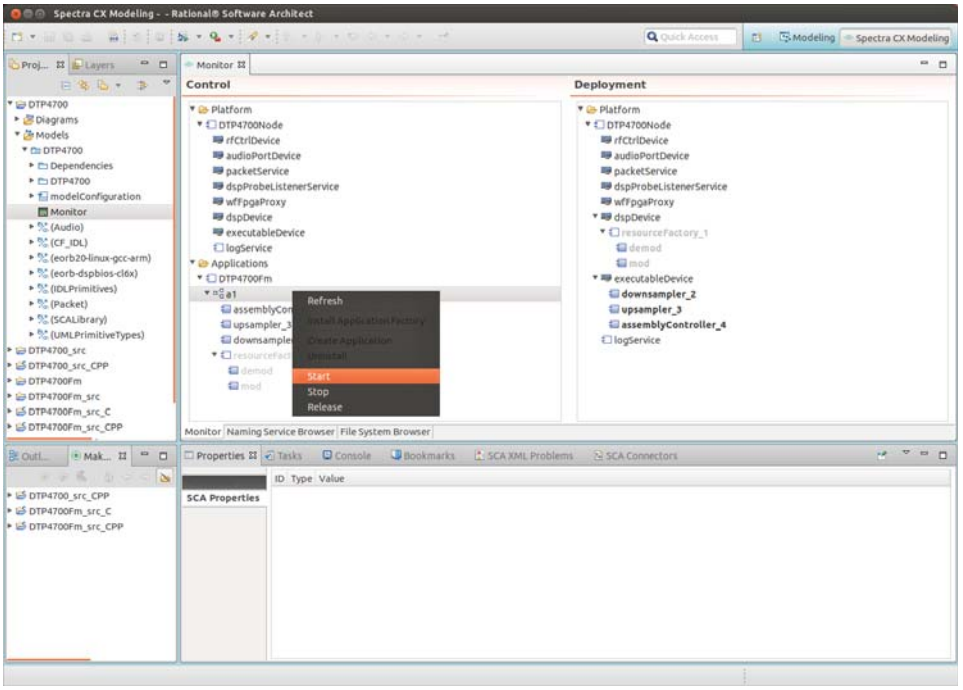

Figure 56 Choose Start from the context menu

i Output from the running application is written to the file `/var/log/dtp.log`.

The application should now be running in receive mode. To enable transmission of FM audio from the Audio In port:

Step 1: Enable `ptt` (push to talk).

Select the audioPortDevice in the Control pane and switch to the Properties view, then select the SCA Properties tab. Select the ptt property and set the value to `true`.

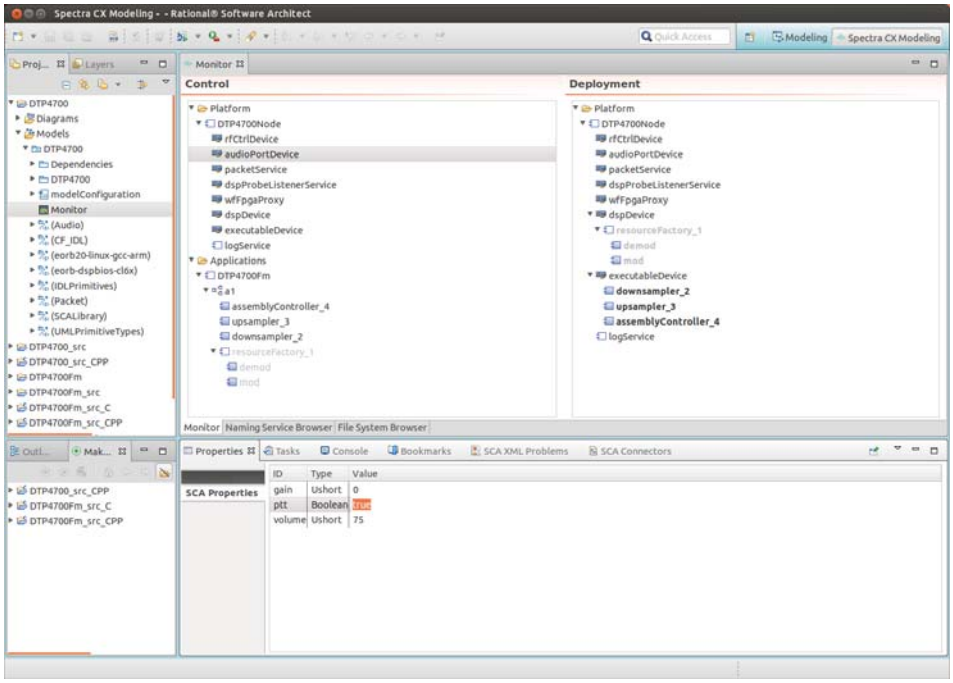

Figure 57 Setting the `ptt` property

To disable transmission of FM audio:

Step 1: Disable `ptt`.

Select the audioPortDevice in the Control pane and switch to the Properties view, then select the SCA Properties tab. Select the ptt property and set the value to `false`.

8.2.10.6 Stopping the FM Waveform

To stop the running application instance:

Step 1: In the Control pane, right-click on the instance a1 of the application DTP4700Fm.

Step 2: Choose **Stop** from the context menu.

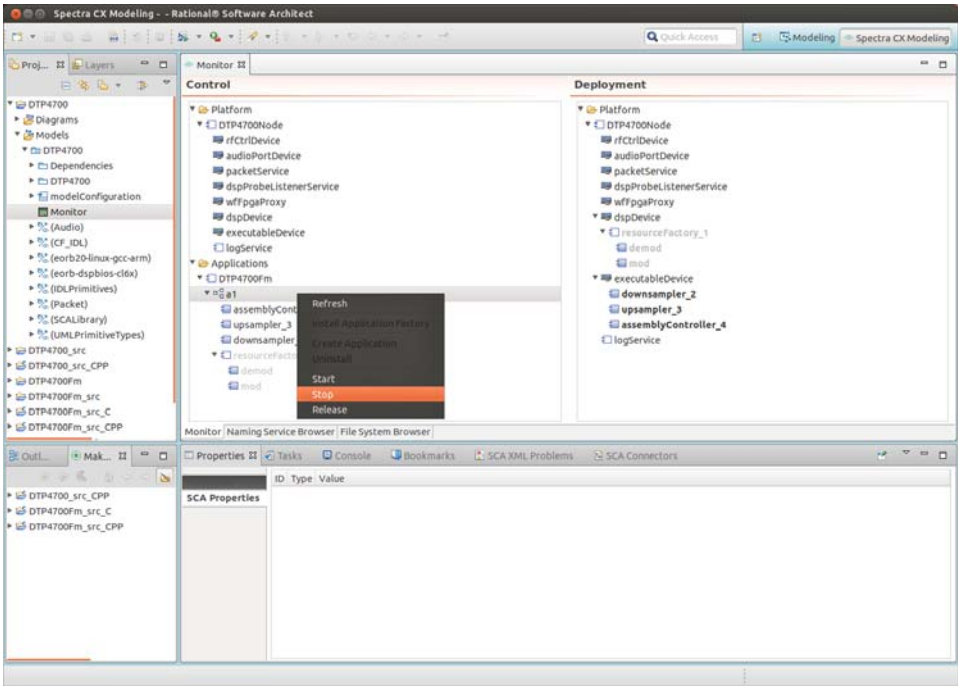

Figure 58 Stopping a running application instance

Stopping the application halts the processing of audio samples.

8.2.10.7 Releasing the FM Waveform

To release the resources of the stopped application instance:

Step 1: In the Control pane, right-click on the instance a1 of the application DTP4700Fm.

Step 2: Choose **Release** from the context menu.

A successful release will tear down the application instance and it will disappear from the Application folder in the Control pane (see *Figure 59* and *Figure 60*). The relevant components also disappear from the Deployment pane.

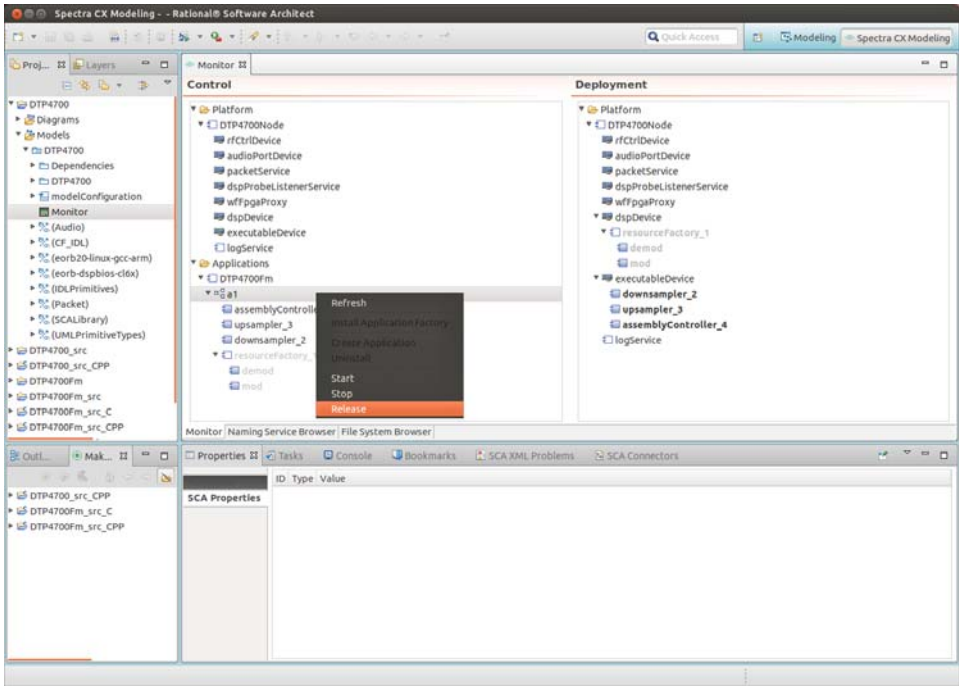

Figure 59 Releasing a stopped application instance

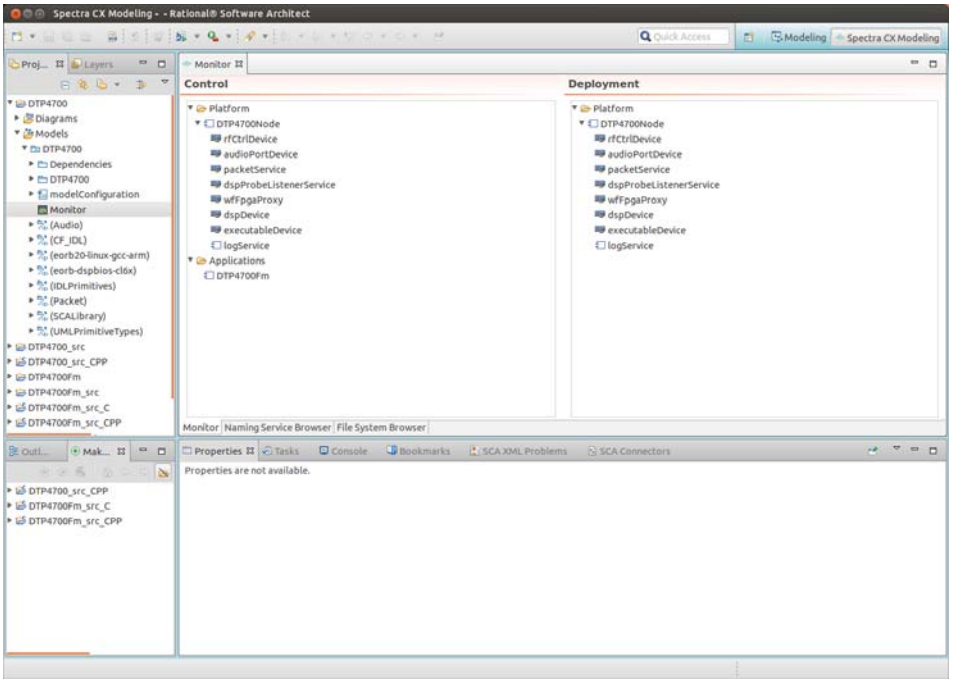

Figure 60 Application instance successfully released

8.2.10.8 Uninstalling the FM Waveform

To uninstall the FM Waveform application:

Step 1: In the Control pane, right-click on the DTP4700Fm application.

Step 2: Choose **Uninstall** from the context menu.

Successfully uninstalling the application removes the waveform from the DTP4700 platform (see *Figure 61* and *Figure 62*).

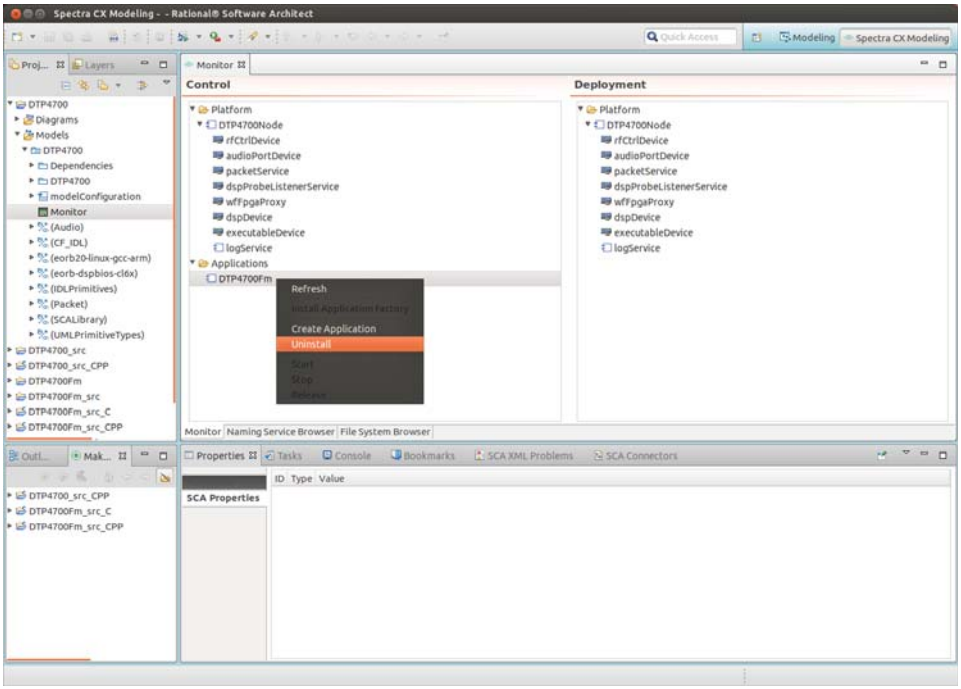

Figure 61 Uninstalling an application

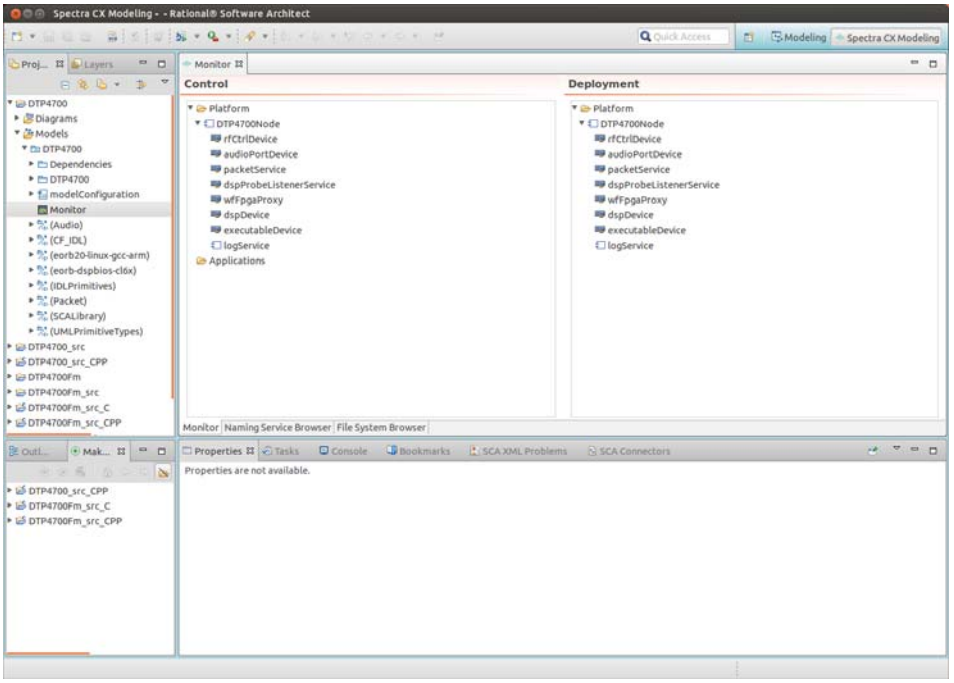

Figure 62 Application successfully uninstalled

8.3 Running the FM Tutorial using cfadmin

The CF platform comes with a console-based utility, `cfadmin`, to control the lifecycle of SCA waveforms. `cfadmin` runs on the Thunder target so that it can interact with all application and platform components.

i This section only gives an example of how to start `cfadmin`; please refer to the *Spectra CF User Guide* for a full description of `cfadmin`, and for step-by-step instructions showing how to use it to install, create, start, stop, release and uninstall waveforms.

8.3.1 Starting cfadmin

Step 1: Open a serial console (`GtkTerm`, telnet or `SSH`) session on the Thunder system and log in as root.

Step 2: Start `cfadmin`, passing it the reference of the Naming Service started by the DomainManager:

```
cfadmin -ORBInitRef
NameService=corbaloc:iiop:1.1@172.31.255.1:2809/NameService
-NAME_BINDING /DTP4700Domain/DomainManager
```

`cfadmin` will display a text menu of options to display and manage various aspects of the domain. Entering 'v' at the menu prompt will display all components that belong to the domain, including application factories, applications (and their resources), device managers, devices and services.

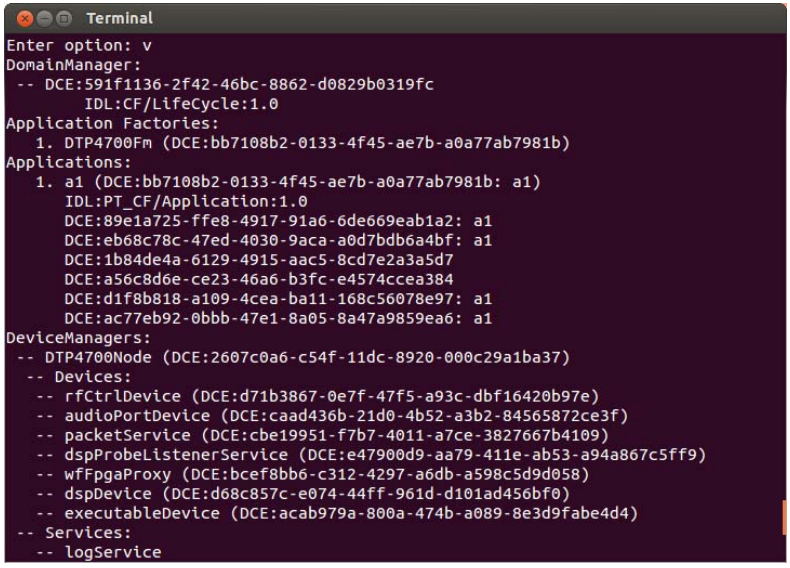

Figure 63 `cfadmin` output confirming application has been created

In *Figure 63* above, the application `a1` shows that it is comprised of six resources. `cfadmin` only displays the instantiation UUIDs of the components as defined in the SAD XML descriptor. An annotated listing of the application components is shown below:

```
DCE:89e1a725-ffe8-4917-91a6-6de669eab1a2: a1 [downsampler]
DCE:eb68c78c-47ed-4030-9aca-a0d7bdb6a4bf: a1 [upsampler]
DCE:1b84de4a-6129-4915-aac5-8cd7e2a3a5d7     [demod]
DCE:a56c8d6e-ce23-46a6-b3fc-e4574ccea384     [mod]
DCE:d1f8b818-a109-4cea-ba11-168c56078e97: a1 [resourceFactory]
DCE:ac77eb92-0bbb-47e1-8a05-8a47a9859ea6: a1 [assemblyController]
```