Keysight VXT PXIe Vector Transceiver

This manual provides documentation for:

Keysight M9410A VXT Vector Transceiver

Keysight M9411A VXT Vector Transceiver

Keysight M9415A VXT Vector Transceiver

Programmer's Guide

Notices

Trademark Acknowledgments

Manual Part Number

M9410-90009

Edition

Edition 2, July 2021

Published in China

Published by:
Keysight Technologies
No 116 Tianfu 4th Street
Chengdu, China 610041

Warranty

THE MATERIAL CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS," AND IS SUBJECT TO BEING CHANGED, WITHOUT NOTICE, IN FUTURE EDITIONS. FURTHER, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, KEYSIGHT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED WITH REGARD TO THIS MANUAL AND ANY INFORMATION CONTAINED HEREIN, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEYSIGHT SHALL NOT BE LIABLE FOR ERRORS OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, USE, OR PERFORMANCE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED HEREIN. SHOULD KEYSIGHT AND THE USER HAVE A SEPARATE WRITTEN AGREEMENT WITH WARRANTY TERMS COVERING THE MATERIAL IN THIS DOCUMENT THAT CONFLICT WITH THESE TERMS, THE WARRANTY TERMS IN THE SEPARATE AGREEMENT WILL CONTROL.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

U.S. Government Rights

The Software is "commercial computer software," as defined by Federal Acquisition Regulation ("FAR") 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement ("DFARS") 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at http://www.keysight.com/find/sweula The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data.

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood and met.

WARNING

A **WARNING** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

Where to Find the Latest Information

Documentation is updated periodically. For the latest information about these products, including instrument software upgrades, application information, and product information, browse to one of the following URLs, according to the name of your product:

http://www.keysight.com/find/m9410a

http://www.keysight.com/find/m9411a

http://www.keysight.com/find/m9415a

To receive the latest updates by email, subscribe to Keysight Email Updates at the following URL:

http://www.keysight.com/find/emailupdates

Information on preventing instrument damage can be found at:

http://www.keysight.com/find/PreventingInstrumentRepair

Is your product software up-to-date?

Periodically, Keysight releases software updates to fix known defects and incorporate product enhancements. To search for software updates for your product, go to the Keysight Technical Support website at:

http://www.keysight.com/find/techsupport

1 Start Programming with IVI Driver

What You Will Learn In This Guide 10
Related Website 11
Related Documentation 12
Overall Process Flow 13
Preparation Before Programming 14
Hardware Installation 14
Software Installation 15
Function Verification 16

2 APIs Introduction

IVI Compliant or IVI Class Compliant 20
IVI Driver Types 21
IVI Driver Hierarchy 23
Instrument-Specific Hierarchies for VXT 24
When Using Visual Studio 25
Naming Conventions Used to Program IVI Drivers 26
General IVI Naming Conventions 26
IVI.NET Naming Conventions 26

3 Creating a Project with IVI.NET Using C#

What you will learn in this chapter 27
Example 1: CW Signal Power Test 28
Step 1 - Create a Console Application 29
Step 2 - Add References 29
Step 3 - Add Using Statements 30
Step 4 - Create and Initialize the Driver Instances 31
Step 5 - Write the Program 33
Step 6 - Close the Driver 34
Step 7 - Build and Run a Complete Program 35
Example 2: Source - Generate LTE FDD Signal 36
Write the Measurement Program 37

- Commands Summary 37
- Get the Measurement Result 38

Example 3: Start a X-Series Application Display 39
- Initialize Instance and Turn on Application UI 40
- Setup SCPI Programming Environment 40
- Set Receiver to Observe Signal 41
- Get the Measurement Result 41
- Basic Concepts: Two VXT Control Method 42

Example 4: Channel Power Acquisition 44
- Write the Measurement Program 45
- Get the Measurement Result 47
- Basic Concepts: 4 Receiver Acquisition Mode 47

Example 5: Spectrum Acquisition 48
- Set VXT Receiver to Test Spectrum Data 49
- Commands Summary 50
- Get the Measurement Result 50

Example 6: FFT Acquisition 51
- Set VXT Receiver 52
- Get the Measurement Result 53

Example 7: IQ Acquisition 54
- Write the Measurement Program 55
- Get the Measurement Result 56

Example 8: Power Servo Measurement 57
- Write the Measurement Program 58
- Commands Summary 60
- Get the Measurement Result 60

Example 9: Harmonics Measurement 61
- Write the Measurement Program 62
- Commands Summary 63
- Get the Measurement Result 63

Example 10: ACPR Test 64
- Write the Measurement Program 65

Commands Summary 66
Get the Measurement Result 67
Example 11: Combined WCDMA Power Servo and ACPR Measurement 68
Example Program – Pseudo – code 68
Source Code 69
Get the Measurement Result 74

Contents

1 Start Programming with IVI Driver

This programmer's guide is intended for individuals who write and run programs to control test-and-measurement instruments. Specifically, in this programmer's guide, you will learn how to use Visual Studio 2015 with the .NET Framework to write Console applications based on IVI.NET driver in Visual C#. Knowledge of Visual Studio 2015 with the .NET Framework and knowledge of the programming syntax for Visual C# is required.

What You Will Learn In This Guide

Our basic user programming model uses the IVI.NET driver directly and allows customer code to:

- Access the IVI.NET driver at the lowest level
- Access IQ Acquisition Mode, Power Acquisition Mode, FFT Acquisition Mode, and Spectrum Acquisition Mode
- Control the Keysight M9410A/M9411A/M9415A VXT Vector Transceiver while performing measurements
- Generate waveforms created by Signal Studio software (licenses are required)

This guide provides the example programs below for your further use with the VXT transceiver:

- Example Program 1: CW Signal Power Test
- Example Program 2: Source - Play Waveform
- Example Program 3: CW Spectrum UI
- Example Program 4: Channel Power Acquisition
- Example Program 5: Spectrum Acquisition
- Example Program 6: FFT Acquisition
- Example Program 7: IQ Acquisition
- Example Program 8: Power Servo
- Example Program 9: Harmonics Test
- Example Program 10: ACPR Test
- Example Program 11: Combined Power Servo and ACPR Measurement

Related Website

- Keysight PXIe and AXIe Modular Products
- Keysight IVI Drivers & Components Downloads
- Keysight I/O Libraries Suite
- Keysight GPIB, USB, & Instrument Control Products
- Keysight VEE Pro
- Keysight Technical Support, Manuals, & Downloads
- Contact Keysight
- IVI Foundation
- MSDN Online

Related Documentation

To access documentation related to the Keysight M9410A/M9411A/M9415A VXT Vector Transceiver Programmer's Guide, use one of the following methods:

- If the product software is installed on your PC, the related documents are also available in the software installation directory.

Table 1-1 Related Documentation

Document	Description	Format
Getting Started Guide	Includes procedures to help you to unpack, inspect, install (software and hardware), perform instrument connections, and troubleshoot your product.	PDF
IVI Driver Reference (Help System)	Provides detailed documentation of the IVI.NET and IVI-C driver API functions, as well as information to help you get started with using the IVI drivers in your application development environment.	CHM (Microsoft Help Format)
X-series Applications Programmer's Guide	Provides basic description about how to program VXT using SCPI commands, and explains how to use the programming documentation.	PDF
User's and Programmer's Reference	Describes the SCPI commands supported by the VXT	CHM (Microsoft Help Format)

- To find the very latest versions of the user documentation, go to the product website (www.keysight.com/find/vxt) and download the files from the Manual support page (go to Resource Center > Document Library > Manuals)

Overall Process Flow

Perform the following steps:

1. Write source code using Microsoft Visual Studio 2015 with .NET Visual C# running on Windows 10.
2. Compile source code using the .NET Framework Library.
3. Produce an Assembly.exe file – you can run this file directly from Microsoft Windows without any other programs.
   - When using the Visual Studio Integrated Development Environment (IDE), the Console Applications you write are stored in conceptual containers called Solutions and Projects.
   - You can view and access Solution and Projects using the Solution Explorer window (View > Solution Explorer).

Preparation Before Programming

If you want to program VXT module to perform measurement, you need to have the following hardwares and softwares:

1. VXT M9410A/M9411A/M9415A modular
2. Chassis (such as Keysight M9018B or M9019A)
3. Controller (such as Keysight M9037A)
4. Reference (such as Keysight M9300A)
5. VXT software
6. M9300A soft front panel
7. IO Libraries Suite (Keysight Connection Expert)
8. Visual Studio (C# or C++ etc) /Labview
9. Window .NET Frameworks Version 4.5.2 or higher version

Hardware Installation

You need install all the listed modules into the chassis as first step of the whole configuration.

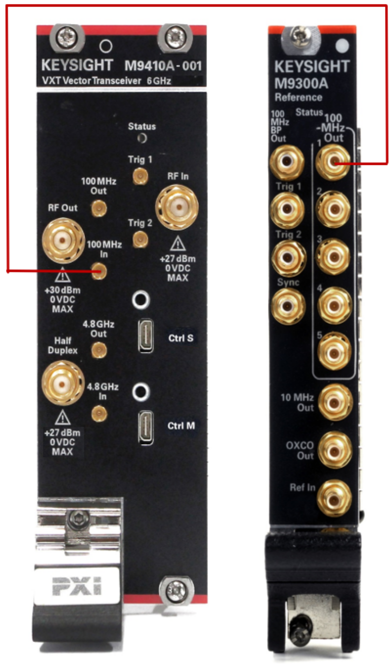
1. Unpack and inspect all hardware.
2. Verify the shipment contents.
3. Install the modules and make cable connections. For detailed procedures, please refer to M9410A/M9411A/M9415A Getting Started Guide.

M9037A Controller

M9410A/M9411A /M9415A VXT

M9300A Reference

4. Connect VXT 100 MHz In port and M9300A 100 MHz Out port with a MMPX male to BNC male cable, such as Keysight Y1815A.



Software Installation

You need install the following softwares before programming with VXT:

1. Install Microsoft Visual Studio with .NET Visual C# running on Windows 7.
2. Install Keysight IO Libraries Suite, this installation includes Keysight Connection Expert.
3. Install the VXT software, Version 16.57 or newer. Driver software includes all IVI.NET and IVI-C Drivers and documentation. All of these items may be downloaded from the Keysight VXT product website.
4. Install the VXT licenses, if you purchased. Please refer to VXT Getting Started Guide for further information.
5. Install the M9300A software. Please refer to M9300A Startup Guide for further information.

   The M9300A PXIe Reference must be included as part of the M941xA configurations. The M9300A PXIe Reference must be initialized first so that the other configurations that depend on the reference signal get the signal they are expecting. If the configuration of modules that is initialized first does not include the M9300A PXIe Reference, unlock errors will occur.

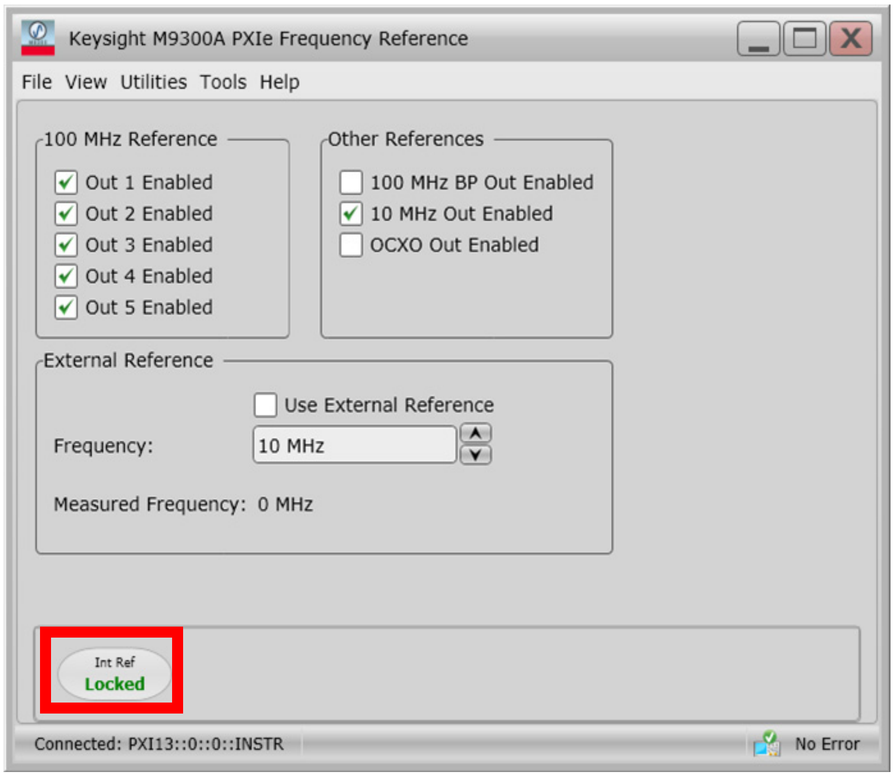
   Once the software and hardware are installed and Self-Test has been performed, they are ready to be programmatically controlled.

Function Verification

To make sure all the hardwares and softwares are ready for your programming, please perform the following steps to generate a CW signal with VXT:

1. Connect VXT RF Output and RF Input with a SMA cable.

2. Power on the chassis and run M9300A software. The software window will be pop up as below. The icon "Int Ref Locked" on the lower left corner indicates the software runs properly.



3. Run VXT software by click *LaunchModularTRX.exe*. Please refer to *VXT Getting Started Guide* for the detailed procedure.

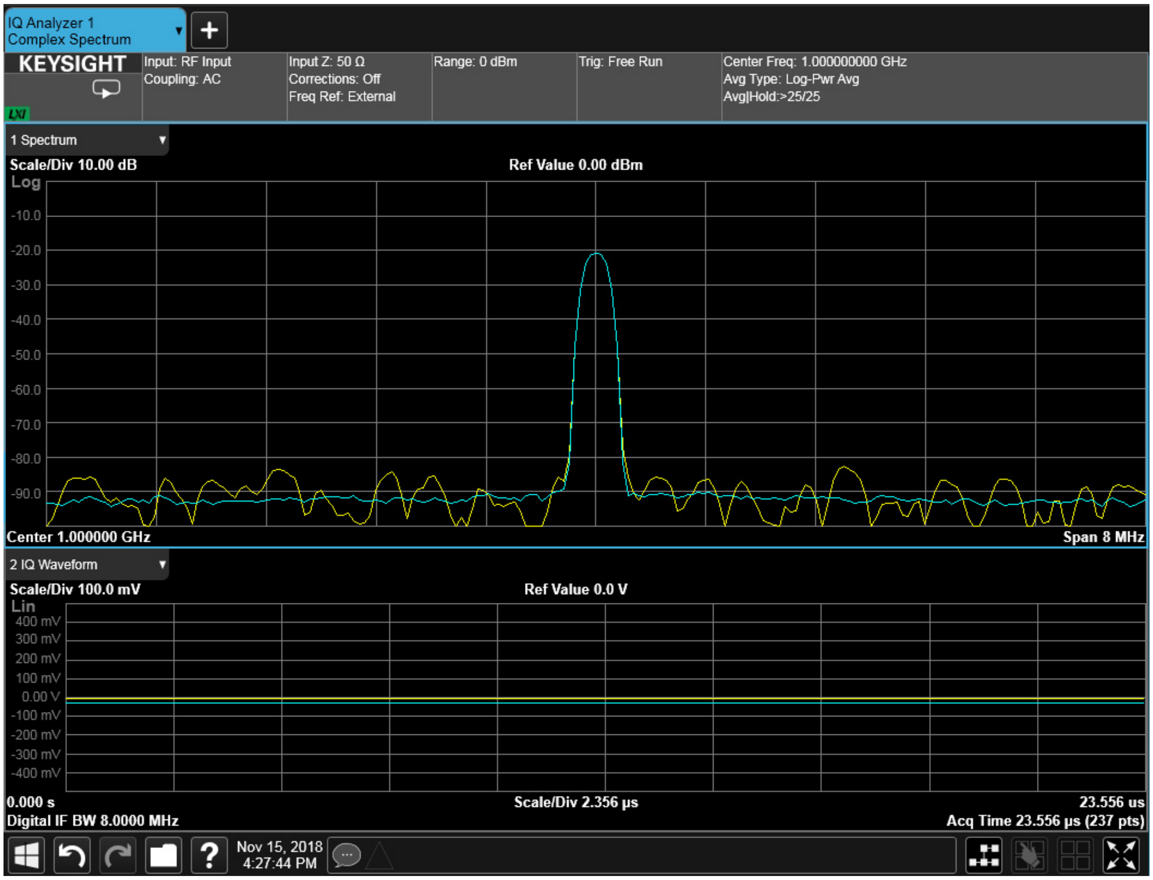
4. Set VXT source to generate a CW signal and use VXT receiver to observe this signal.

    1. Press Frequency > Input/Output > Source Amplitude > - 20 dBm to set the signal amplitude to -20 dBm

    2. Press Frequency > 1 GHz to set the signal frequency to 1 GHz.

    3. Press RF Output On to turn on the source output

4. Press Input/Output > Frequency > Center Freq > 1 GHz to set the receiver center frequency to 1 GHz.



If you observe the CW signal as figure above, it indicates the VXT is ready for your programming, VXT supports multiple programming platform, such as Visual C#, Visual Basic .Net, Visual C++, Keysight VEE pro, Labview and MATLAB. In this guide, all the programming examples are programmed with Visual C#.

2 APIs Introduction

This chapter describes the Application Programming Interfaces (APIs) for the Keysight VXT vector transceiver.

The following IVI driver terminology may be used when describing the Application Programming Interfaces (APIs) for the VXT Vector Transceiver.

IVI[Interchangeable Virtual Instruments] - a standard instrument driver model defined by the IVI Foundation that enables engineers to exchange instruments made by different manufacturers without rewriting their code.

Currently, there are 13 IVI Instrument Classes defined by the IVI Foundation. The VXT Vector Transceiver do not belong to any of these 13 IVI Instrument Classes and are therefore described as "NoClass" modules.

- DC Power Supply
- AC Power Supply
- DMM
- Function Generator
- Oscilloscope
- Power Meter
- RF Signal Generator
- Spectrum Analyzer
- Upconverter
- Downconverter
- Digitizer
- Counter/Timer

IVI Compliant or IVI Class Compliant

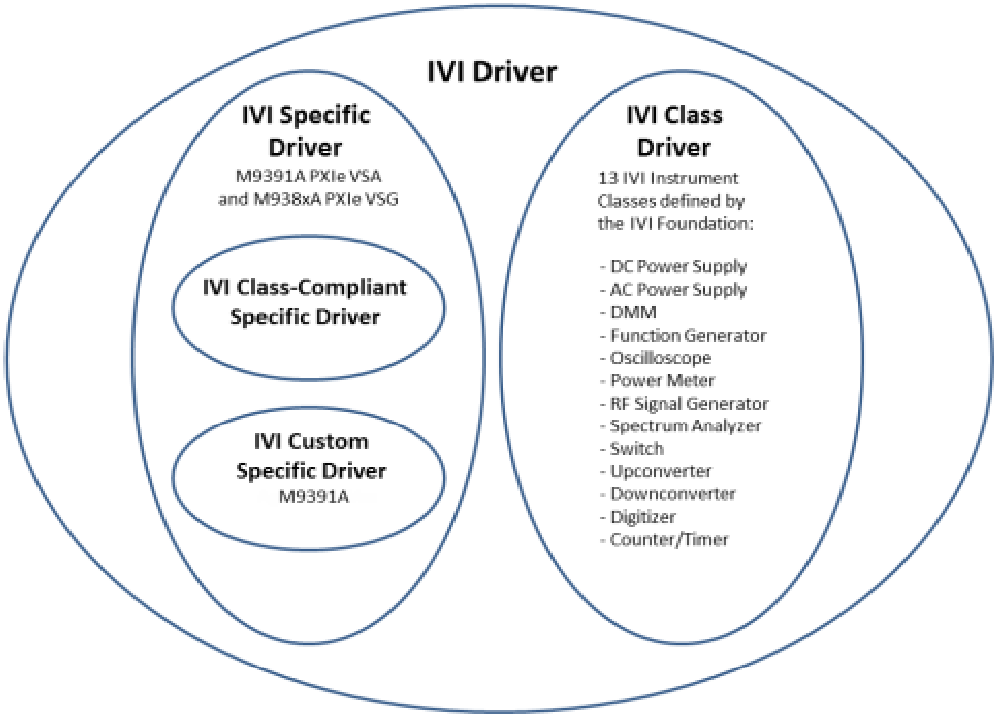
The VXT Vector Transceiver is IVI Compliant, but not IVI Class Compliant; none of these belongs to one of the 13 IVI Instrument Classes defined by the IVI Foundation.

- IVI Compliant - means that the IVI driver follows architectural specification for these categories:
  - Installation
  - Inherent Capabilities
  - Cross Class Capabilities
  - Custom Instrument API
- IVI Class Compliant - means that the IVI driver follows architectural specification for these categories:
  - If an instrument is IVI Class Compliant, it is also IVI compliant
  - Provides one of the 13 IVI Instrument Class APIs is in addtion to a Custom API
  - Custom API may be omitted (unusual)
  - Simplifies exchanging instruments

IVI Driver Types

There are several types of IVI drivers as listed below:

Figure 2-1 *IVI Driver Types*



- IVI Driver
  - Implements the Inherent Capabilities Specification
  - Complies with all of the architecture specifications
  - May or may not comply with one of the 13 IVI Instrument Classes
  - Is either an IVI Specific Driver or an IVI Class Driver
- IVI Class Driver
  - Is an IVI Driver needed only for interchangeability in IVI-C environments
  - The IVI Class may be IVI-defined or customer-defined
- IVI Specific Driver
  - Is an IVI Driver needed only for interchangeability in IVI-C environments
  - The IVI Class may be IVI-defined or customer-defined
- IVI Class-Compliant Specific Driver
  - IVI Specific Driver that complies with one (or more) of the IVI defined class specifications
  - Used when hardware independence is desired

- IVI Custom Specific Driver
  - Is an IVI Specific Driver that is not compliant with any one of the 13 IVI defined class specifications
  - Used when hardware independence is desired

NOTE

This release is not binary compatible with prior releases of the IVI-C driver. Programs using the C/C++ IVI-C driver must be recompiled for this version of the driver. Similarly, programs compiled with this version of the driver will not be compatible with older versions of the IVI-C driver. This incompatibility is due to renumbering of attribute constants defined in the KtM941x.h include file.

IVI Driver Hierarchy

When writing programs, you will be using the interfaces (APIs) available to the IVI.NET driver.

- The core of every IVI.NET driver is a single object with many interfaces.
- These interfaces are organized into two hierarchies: Class-Compliant Hierarchy and Instrument-Specific Hierarchy – and both include the IIviDriver interfaces.
  - Class-Compliant Hierarchy - Since the VXT Vector Transceiver does not belong to one of the 13 IVI Classes, there is noClass-Compliant Hierarchy in their IVI Driver.
  - Instrument-Specific Hierarchy
    - The VXT Vector Transceiver's instrument-specific hierarchy has IKtM9410 at the root (where KtM9410 is the driver name).
      - IKtM9410 is the root interface and contains references to child interface, which in turn contain references to other child interfaces. Collectively, these interfaces define the Instrument-Specific Hierarchy.
  - The IIviDriver interfaces are incorporated into both hierarchies: Class-Compliant Hierarchy and Instrument-Specific Hierarchy.

    The IIviDriver is the root interface for IVI Inherent Capabilities which are what the IVI Foundation has established ass a set of functions and attributes that all IVI drivers must include - irrespective of which IVI instrument class the driver supports. These common functions and attributes are called IVI inherent capabilities and they are documented in IVI-3.2 -Inherent Capabilities Specification. Drivers that do not support any IVI instrument class such as the VXT Vector Transceiver must still include these IVI inherent capabilities.

    Close

    DriverOperation

    Identity

    Initialized

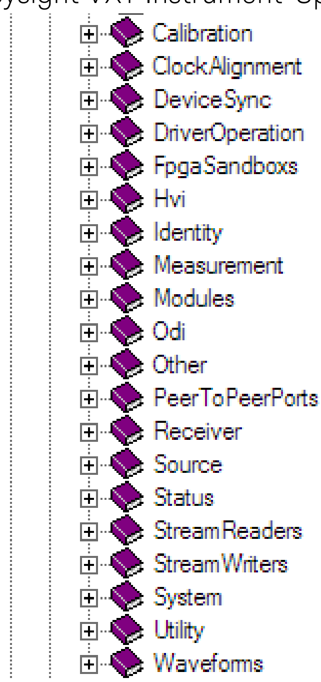
    Utility

Instrument-Specific Hierarchies for VXT

The following table lists the instrument-specific hierarchy interfaces for M941XA VXT Vector Transceiver.

Keysight VXT Instrument-Specific Hierarchy
KtM941x is the driver name
IKtM941x is the root interface

Figure 2-2 Keysight VXT Instrument-Specific Hierarchy



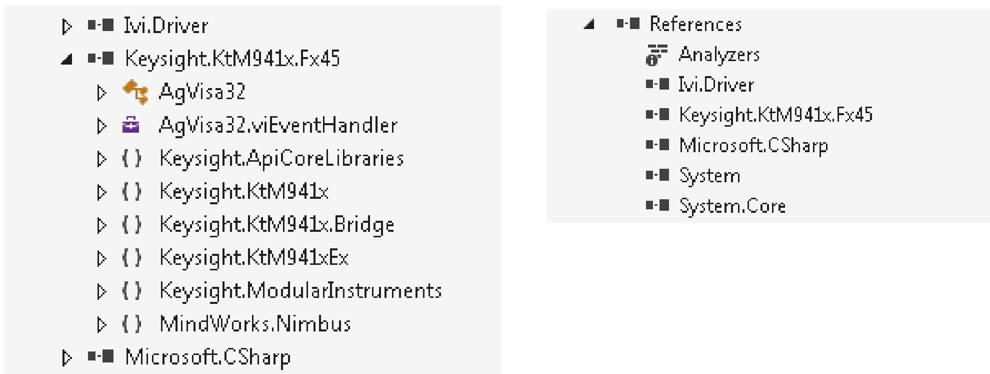
NOTE

All new code being created should use the IKtM941x extended interfaces in place of the IKtM9410 interfaces. New functionalities have been added to the IKtM941x extended interfaces. These new functionalities were not available in the original IKtM9410 interfaces, and have been left unchanged to support previously written code; this helps support backward code compatibility.

When Using Visual Studio

To view interfaces available in VXT, click KtM941x library file, in the References folder, from the Solution Explorer window and select View in Object Browser.

Figure 2-3 Keysight VXT Instrument-Specific Hierarchy



Naming Conventions Used to Program IVI Drivers

General IVI Naming Conventions

- All instrument class names start with "IVi"

  Example: IviScope, IviDmm

- Function names

  One or more words use PascalCasing

  First word should be a verb

IVI.NET Naming Conventions

- Interface naming

  Class compliant: Starts with "IIvi"

  I<ClassNaming>

  Example:IIviScope, IIviDmm

- Sub-interfaces add words to the base name that match the C hierarchy as close as possible

  Example: IIviFgenArbitrary, IIviFgenArbitraryWaveform

- Enum values don’t end in "Enum" but use the last word to differentiate

  Example: IviScopeTriggerCouplingAC AND IviScopeTriggercouplingDC

3 Creating a Project with IVI.NET Using C#

What you will learn in this chapter

This tutorial will walk through the various examples to create a console applications using Visual Studio and C#. It demonstrates how to instantiate driver instance, set the resource names and various initialization values, initialize the driver instance, print various driver properties to a console for each driver instance, check drivers for errors and report the errors if any occur, and close both drivers.

The project examples are listed below.

1. CW Signal Power Test
2. Source - Play Waveform
3. Start X Application
4. Channel Power Acquisition
5. Spectrum Acquisition
6. FFT Acquisition
7. IQ Acquisition
8. Power Servo Measurement
9. Harmonics Measurement
10. ACPR Measurement
11. Combined Power Servo and ACPR Measurement

All the example programs above are in the folder below after the VXT software is installed.

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\
Keysight.KtM941x x.x.x\Examples\CSharp

Example 1: CW Signal Power Test

This example introduces the programming procedure to perform a CW signal power test as below with VXT using Visual Studio and C#.

- VXT source outputs a 1 GHz CW signal
- VXT receiver measures this signal power

Figure 3-1 VXT CW Signal Power Test Cable Connection

NOTE Before programming, please connect VXT RF Output to RF Input port.

The programming procedure are listed as 8 steps as below:

Step 1. - Create a "Console Application"

Step 2. - Add References

Step 3. - Add Using Statements

Step 4. - Create an Instance

Step 5. - Initialize the Instance

Step 6. - Write the Program (Create a Signal or Perform a Measurement)

Step 7. - Close the Instance

Step 8. - Build and Run the Program

After the VXT software is installed, you will find the source code as below:

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsCWPowerTest.

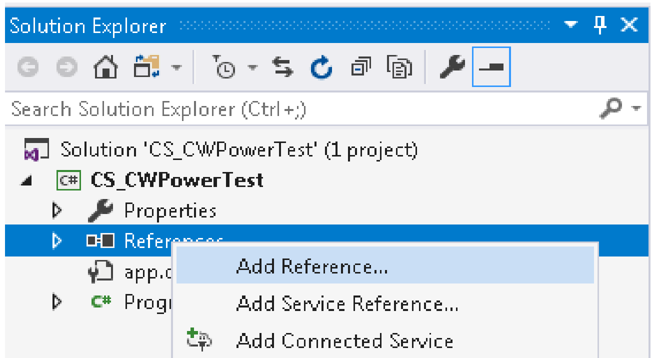
Step 1 – Create a Console Application

1. Launch Visual Studio and create a new Console Application in Visual C# by selecting: **File** > **New** > **Project** and select a Visual C# Console Application.
2. Enter **"CWPowerTest"** as the Name of the project and click OK.

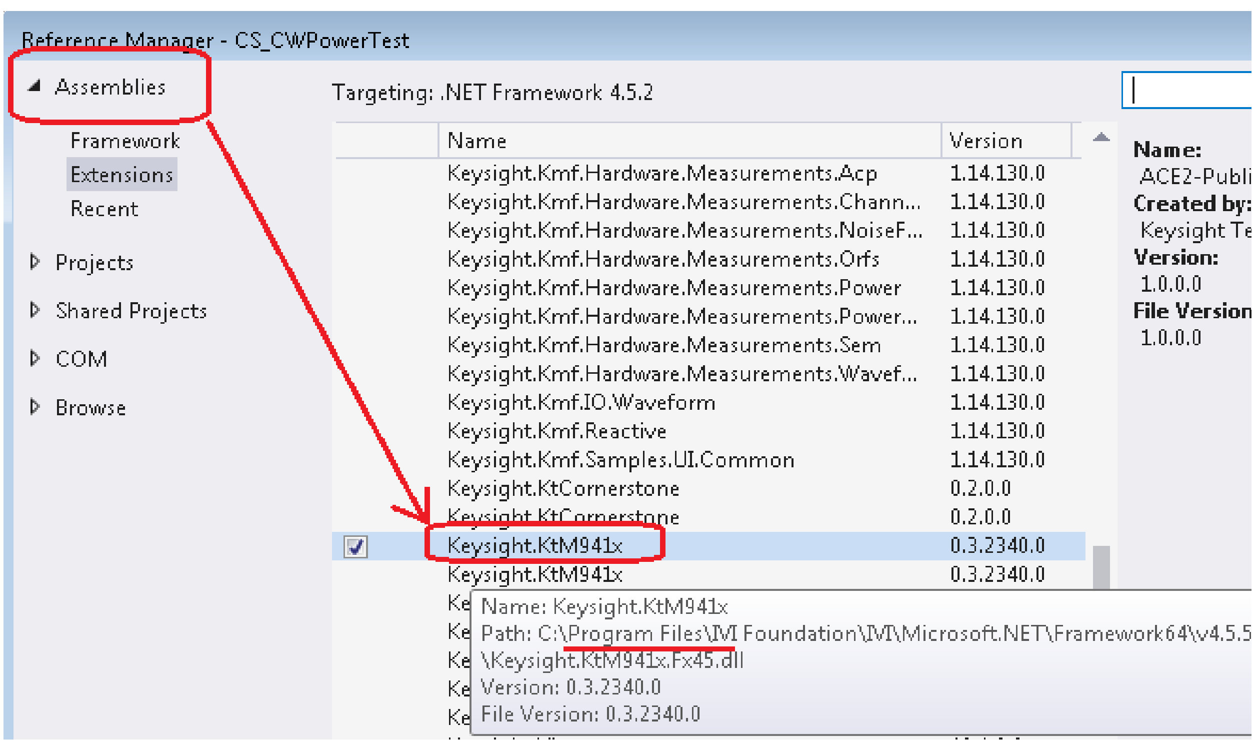
Step 2 – Add References

In order to access the VXT driver interfaces, references to their drivers (DLL) must be created.

1. In Solution Explorer, right-click on References and select Add Reference.



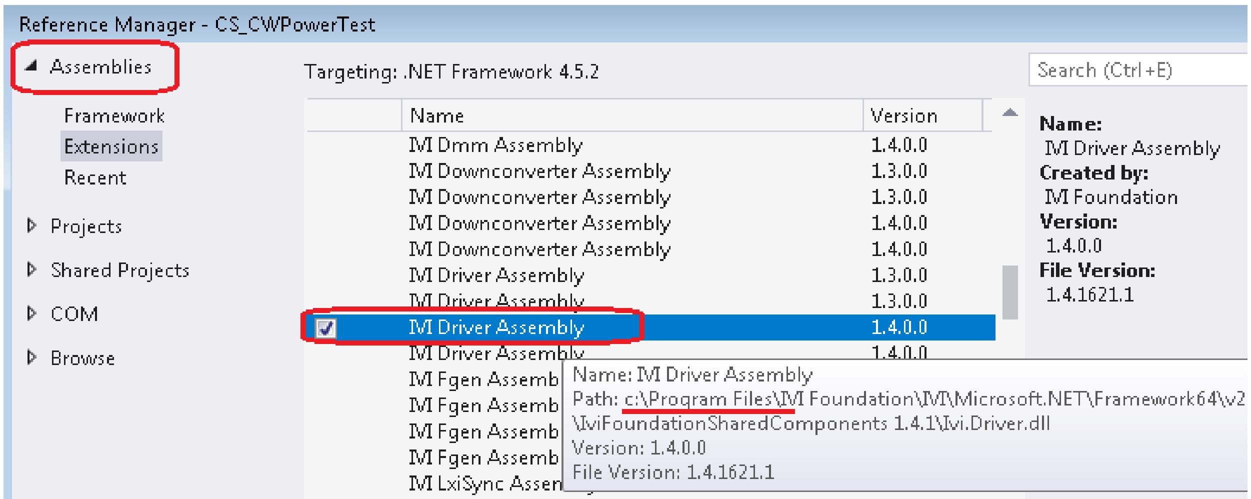
2. From the Reference Manager, select the Assemblies table to find VXT's driver – Keysight.KtM941x. Take note there are two Keysight.KtM941X drivers in the list, one for 64 bits system and the other for X86 system. Choose the 64 bits driver, then press OK to confirm. The 64 bits driver is in C:\Program Files\, while the 32 bits one is in C:\Program Files (x86)\.



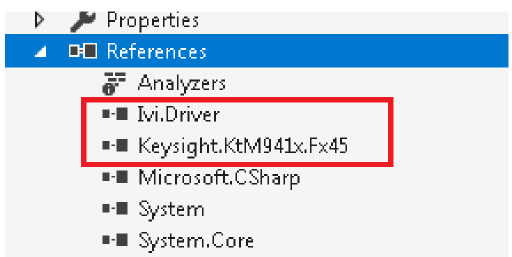
NOTE

If you have not installed the IVI driver for the VXT product (as listed in chapter 1, Before Programming, Install Hardware, Software, and Software Licenses), the IVI drivers will not appear in this list.

3. Select the IVI Driver Assembly 1.4.0.0 or higher version, and press OK to confirm. There are two IVI Driver assembly file for each version, one is for 64 bits system and another is for X86 system. Confirm the version via the file path from the address in the figure below.



4. Now the IVI drivers are referenced and available for your use.



Step 3 - Add Using Statements

To allow your program to access the IVI drivers without specifying full path names of each interface or enum, you need to add using statements to your program.

All data types (interfaces and enums) are contained within namespaces. (A namespace is a hierarchical naming scheme for grouping types into logical categories of related functionality. Design tools, such as Visual Studio, can use namespaces which makes it easier to browse and reference types in your code.)The C# using statement allows the type name to be used directly. Without the using statement, the complete namespace-qualified name must be used. To allow your program to access the IVI driver without having to type the full path of each interface or enum, type the following using statements immediately below the other using statements.

```
using System;
using System.Threading;
using Keysight.KtM941x;

namespace CS_CWPowerTest
{
    /// Keysight VXTII M941xA Driv
```

Step 4 - Create and Initialize the Driver Instances

There are two ways to instantiate (create an instance of) the IVI.NET drivers:

- Direct Instantiation
- COM Session Factory

Since the VXT vector transceiver is considered NoClass module (because they do not belong to one of the 13 IVI Classes), the COM Session Factory is not used to create instances of their IVI.NET drivers. So, VXT vector transceiver IVI.NET driver uses direct instantiation. Because direct instantiation is used, their IVI.NET drivers may not be interchangeable with other modules.

To create driver instances, the new operator is used in C# as below.

```
IKtM941x Driver = new KtM941x();
```

The `Initialize()` method is required when using any IVI driver. It establishes a communication link (an "I/O session") with an instrument and it must be called before the program can do anything with an instrument or work in simulation mode.

The `Initialize()` method has a number of options that can be defined. In this example, we prepare the `Initialize()` method by defining only a few of the parameters, then we call the `Initialize()` method with the parameters below.

To initialize the driver instances, the example code below is used in C#.

```
KtM941x driver = null;

string ResourceName = "PXI21::0::0::INSTR";

bool IdQuery = true;

bool Reset = true;

string OptionString = "QueryInstrStatus=true, Simulate=false,
DriverSetup= Model=M941xA";

driver = new KtM941x(ResourceName, IdQuery, Reset, OptionString);
```

Initialize Parameters

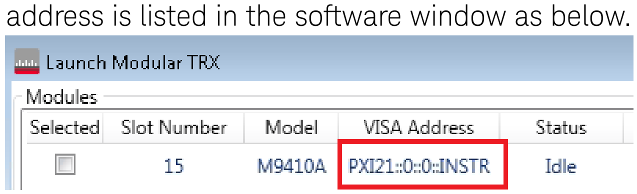
The following tables describes options that are most commonly used to initialize the instance.

Property Type and Example Value	Description of Property
string ResourceName = "PXI13::0::0::INSTR";	VxtResourceName – The driver is typically initialized using a physical resource name descriptor, often a VISA resource descriptor. See the procedure in the Resource Names section in the next page.
bool IdQuery = true;	Setting the ID query to false prevents the driver from verifying that the connected instrument is the one the driver was written for because if IdQuery is set to true, this will query the instrument model and fail initialization if the model is not supported by the driver.
bool Reset = true;	Setting Reset to true instructs the driver to initially reset the instrument.
string OptionString = "QueryInstrStatus=true,	OptionString - Setup the following initialization options:
Simulate=true,	· QueryInstrStatus=true (Specifies whether the IVI specific driver queries the instrumetn status at the end of each user operation.) · Simulate=true (Setting Simulate to true instructs the driver to not to attempt to connect to a physical instrument, but use a simulation of the instrument instead.) · Cache=false (Specifies whether or not to cache the value of properties.) · InterchangeCheck=false (Specifies whether the IVI specific driver performs interchangeability checking.) · RangeCheck=false (Specifies whether the IVI specific driver validates attribute values and function parameters.) · RecordCoercions=false (Specifies whether the IVI specific driver keeps a list of the value coercions it makes for ViInt32 and ViReal64 attributes.)
DriverSetup=";	· DriverSetup= (This is used to specify settings that are supported by the driver, but not defined by IVI. If the Options String parameter (OptionString in this example) contains an assignment for the Driver Setup attribute, the Initialize function assumes that everything following 'DriverSetup=' is part of the assignment.)

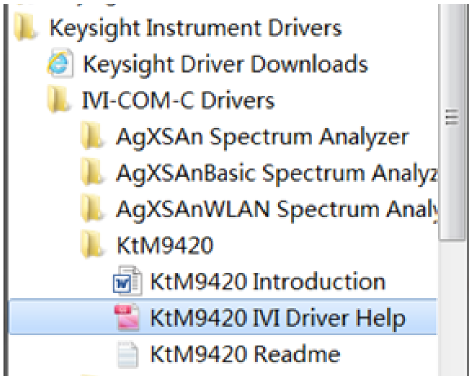
Resource Names

You need to determine the Resource Name address string (VISA address string) that is needed.

- Run VXT software (*LaunchModularTRX.exe*). The VXT modular' VISA address is listed in the software window as below.



Please refer to VXT IVI Driver *Help* (**Start** > **All programs** > **Keysight Instrument Drivers** > **IVI.NET Drivers** > **KtM941x** > **KtM941x IVI Driver Help**) as below for further information.



Step 5 - Write the Program

At this point, you can add program steps that use the driver instances to perform tasks.In this example, perform the following steps:

1. Set VXT source to generate a -10 dBm CW signal at 1 GHz.
2. Set VXT receiver to measure the power of the CW signal.

Set the VXT Source

To set the VXT source to generate a -10 dBm CW signal at 1 GHz, please refer to the example code as below:

```
driver.Source.RF.Frequency = 1e9;
//Set the source's center frequency.
driver.Source.RF.Level = -10;
//Set the source's RF power level.
driver.Source.RF.OutputPort = Port.RFOutput;//Select source output port
//Select the source output port
driver.Source.RF.OutputEnable = true;
//Enable output.
driver.Apply();
//Apply the above setting to VXT source's hardware.
```

For more APIs about VXT source frequency settings, please refer to the VXT IVI driver help as below.

Set the VXT Receiver

To measure the channel power in a bandwidth, please refer to the example code below:

```
driver.Receiver.RF.Frequency = 1e9; //Set Receiver's Center Freq
driver.Receiver.RF.Power = -5;
//the Receiver.RF.Power should be set equal to or little bigger than target
test value, to get exact test result
driver.Receiver.RF.PeakerToAverage = 3;
//Set Receiver's Peak to Average value. It's a important setting for digital
modulation signal test.

driver.Receiver.RF.InputPort = Port.RFInput;//Set the RF input port
driver.Receiver.AcquisitionMode = AcquisitionMode.Power; //Choose Power
Acquisition Mode.
driver.PowerAcquisition.Bandwidth = 1e6;
driver.PowerAcquisition.Duration = 0.02;
driver.Receiver.PowerAcquisition.ChannelFilter.Shape =
ChannelFilterShape.None;
driver.PowerAcquisition.ChannelFilter.Alpha = 0.5;
driver.PowerAcquisition.ChannelFilter.Bandwidth = 1e5;//Set the channel
bandwidth.

driver.Apply();//Apply the above setting to VXT receiver's hardware.
driver.Arm();//Start the M941xA's receiver's measurement

driver.Receiver.PowerAcquisition.ReadPower(CAPTURE_ID, PowerUnits.dBm, out
power, out overloaded); //Read the power measurement result.
```

For more example codes of frequently used measurement cases, please refer to the other examples introduced in this chapter.

Step 6 – Close the Driver

Calling `Close()` at the end of the program is required by the IVI specification when using any IVI driver.

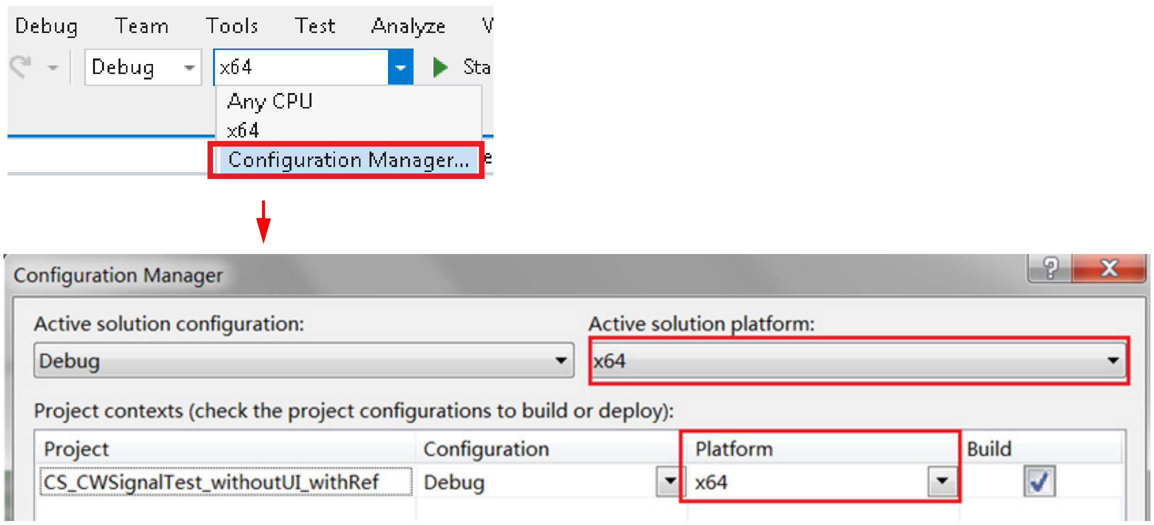
```
driver.Close();
```

`Close()` may be the most commonly missed step when using an IVI driver. Failing to do this could mean that system resources are not freed up and your program may behave unexpectedly on subsequent executions.

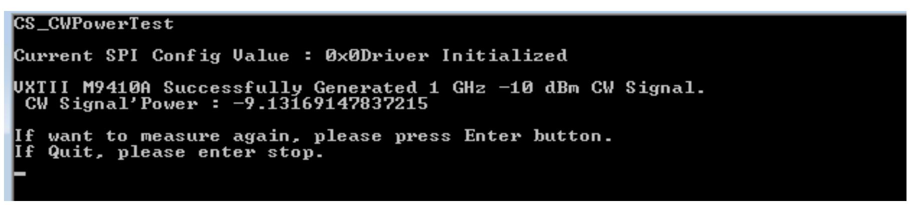
Step 7 – Build and Run a Complete Program

Build your console application and run it to verify it works properly.

1. Open the solution file CWSignalTest.sln in Visual Studio.
2. Set the appropriate platform target. If the installed VXT software is 64-bit, you need to set the active solution platform as X64 in configuration manager.



3. Choose Project > CWSignalTest Properties and select Build/Rebuild Solution. And the program file will be built out.
4. Run the program and you will get the test result as below.



Before running the program, please make sure the M9300A reference software is turned on.

It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

For the most of the VXT programs, the step 1, 2, 3, 6, 7 are same. The only difference is in step 4, 5, you need program your own code.

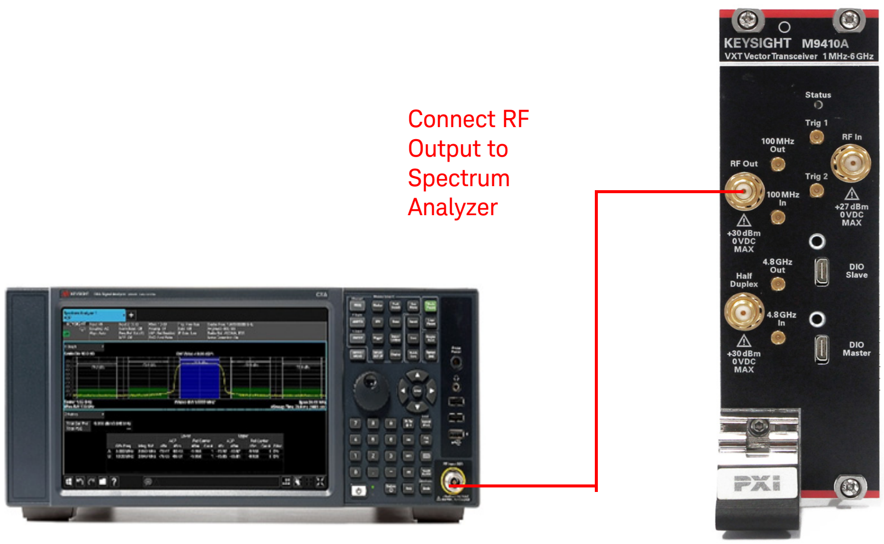
From the example 2 to 10, we will just focus on step 5 - **Write the Program**.

Example 2: Source - Generate LTE FDD Signal

This example introduces the programming procedure to output a LTE FDD signal with M9410A/M9411A/M9415A source using Visual Studio and C#.

NOTE **N7624B Signal Studio for LTE/LTE-FDD is needed to play a LTE FDD signal with VXT product.**

Figure 3-2 VXT Source Play Waveform Cable Connection



NOTE **Before programming, please connect VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port.**

The programming procedure are listed as 8 steps as below:

Step 1. - Create your project with Visual C#

Step 2. - Add References

Step 3. - Add Using Statements

Step 4. - Create and Initialize the Instance

Step 5. - Write the Program (Generate a LTE FDD signal with M941xA source)

Step 6. - Close the Instance

Step 7. - Build and Run the Program

For step 1, 2, 3, 4, 6, 7, pleas refer to example 1 as those steps are similar. This section will only introduce the example code for step 5.

After the VXT software is installed, you will find the source code as below:

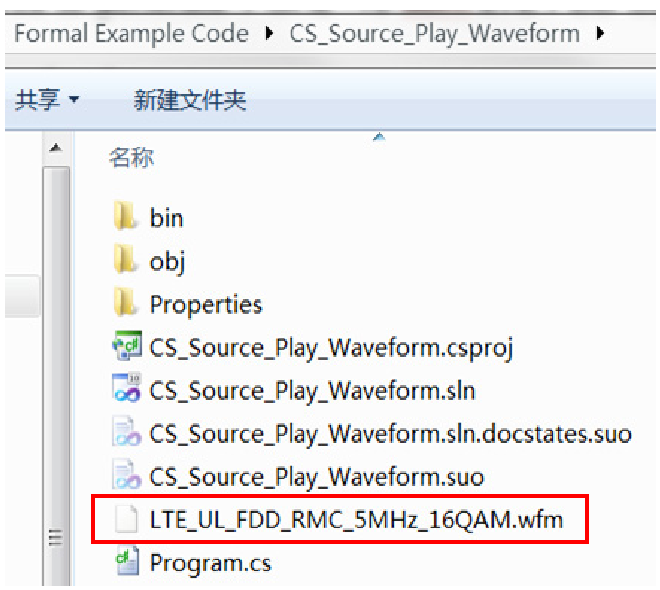
C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsSource_Play_Waveform.

Write the Measurement Program

To output a LTE FDD signal with M9410A/M9411A/M9415A source, please refer to the example code as below:

```
string filePath = "C:\\Waveform";
string fileName = "LTE_UL_FDD_RMC_5MHz_16QAM.wfm";
driver.Source.LoadWaveform(filePath, fileName); //load the wave form

driver.Source.RF.Frequency = 1e9;//Set the source center frequency. In this
code it's set to 1e9Hz.
driver.Source.RF.Level = -5; //Set the source's RF power level.
driver.Source.RF.OutputPort = Port.RFOutput; //Select source output port
driver.Source.RF.OutputEnabled = true; //Enable output.
driver.Source.Modulation.PlayArb(fileName, startEvent: StartEvent.Immediate);
driver.Source.Modulation.Enabled = true;

driver.Apply(); //Apply the above setting to VXT source's hardware.
```

The waveform file "*LTE_UL_FDD_RMC_5MHz_16QAM.wfm*" used in this example code is attached in the project file.



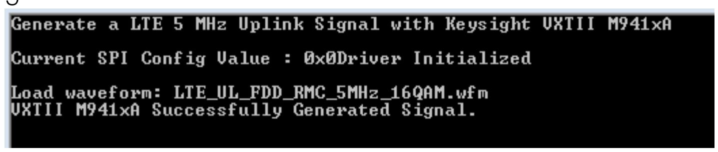
Commands Summary

- `Driver.Apply()` method is used to update all the parameter setting, for VXT source and receiver. It is a frequently used method.
- The methods of `Driver.Source.RF` are used to set the basic RF parameters, such as output signal freq, level, output port. If you don't play any waveform, it will generate a CW signal as example 1 does.

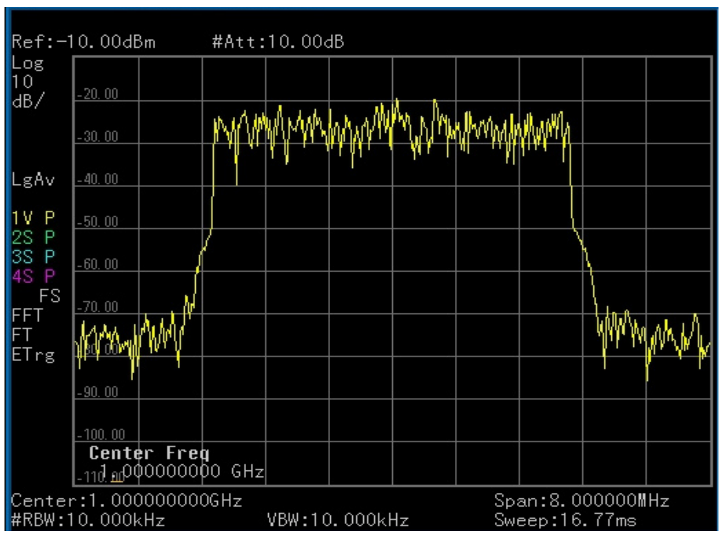
- `Driver.Source.LoadWaveform()` is used to load waveform to VXT memory, It allows you to load multiple waveform files into memory at the same time.
- `Driver.Source.Modulation.PlayArb` is used to choose the waveform to play. VXT supports loading multiple waveform into memory as generating digital demodulation signal.

Get the Measurement Result

Refer to the process of step 7 in example 1 to build and run your program to get the result as below.



```
Generate a LTE 5 MHz Uplink Signal with Keysight VXTII M941xA

Current SPI Config Value : 0x0Driver Initialized

Load waveform: LTE_UL_FDD_RMC_5MHz_16QAM.wfm
VXTII M941xA Successfully Generated Signal.
```



It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

Example 3: Start a X-Series Application Display

This example introduces the programming procedure to display a spectrum in X series spectrum UI with M941xA.

- VXT source outputs a 1 GHz CW signal
- Turn on the X series spectrum analyzer application display and use SCPI command to set the receiver display the spectrum

Figure 3-3 Example 3 - Cable Connection

NOTE Before programming, please connect VXT RF Output to RF Input port and VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port.

The programming procedure are listed as 8 steps as below:

Step 1. - Create your project with Visual C#

Step 2. - Add references

Step 3. - Add using statements

Step 4. - Create and initialize the Instance

Step 5. - Set source to generate CW signal

Step 6. - Setup SCPI programming environment

Step 7. - Set receiver to observe the CW signal with X-series spectrum UI

Step 8. - Close the Instance

Step 9. - Build and Run the Program

For step 1, 2, 3, 4,5, 8, 9, pleas refer to example 1 as those steps are similar. This section will only introduce the example code for step 6 and 7.

After the VXT software is installed, you will find the source code as below:

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsStart_X_APP.

Initialize Instance and Turn on Application UI

To initialize the driver instance and turn on Application UI, please refer to the example code as below:

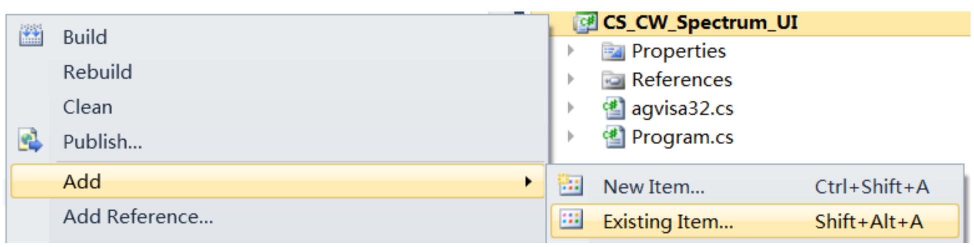
```
string ResourceName = "PXI0::21-0.0::INSTR";
bool IdQuery = true;
bool Reset = true;
string OptionString = "QueryInstrStatus=true, Simulate=false, DriverSetup=
Model=M941xA, TouchGuiStart=true";
driver = new KtM941x(ResourceName, IdQuery, Reset, OptionString);
```

`AppStart=true` is used to turn on the VXT application UI display. If you do not need the UI display, just delete the `AppStart = true` from the Option String.

Setup SCPI Programming Environment

To setup SCPI programming environment, please refer to the procedure below:

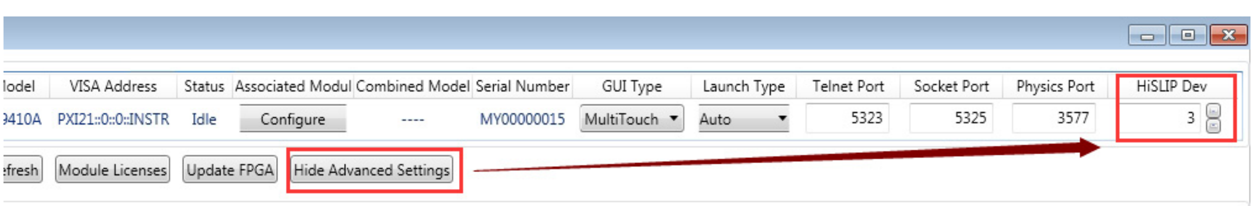
1. Install Keysight IO Libraries Suite.
2. Add file "agvisa32.cs" into your project. Select Add > Existing Item in Visual Studio as below. The file "agvisa32.cs" is located at C:\Program Files (x86)\IVI Foundation\VISA\WinNT\agvisa\include



3. Add the code below to your project to enable VISA connection with X series application.

```
int rm = 0; int xApp;
AgVisa32.viOpenDefaultRM(out rm);
AgVisa32.viOpen(rm, "TCPIP0::localhost::hislip3::INSTR", 0, 0, out xApp);
AgVisa32.viSetAttribute(xApp, AgVisa32.VI_ATTR_TMO_VALUE, 10000);
AgVisa32.viPrintf(xApp, ":SYST:ERR:VERB ON;\n"); //clear the system error
information
```

The VISA connection needs hislip LAN address. The "TCPIP0::localhost::hislip3::INSTR" used in viOpen method is a hislip address. Please run *LaunchModularTRX.exe* to get the hislip address as below.



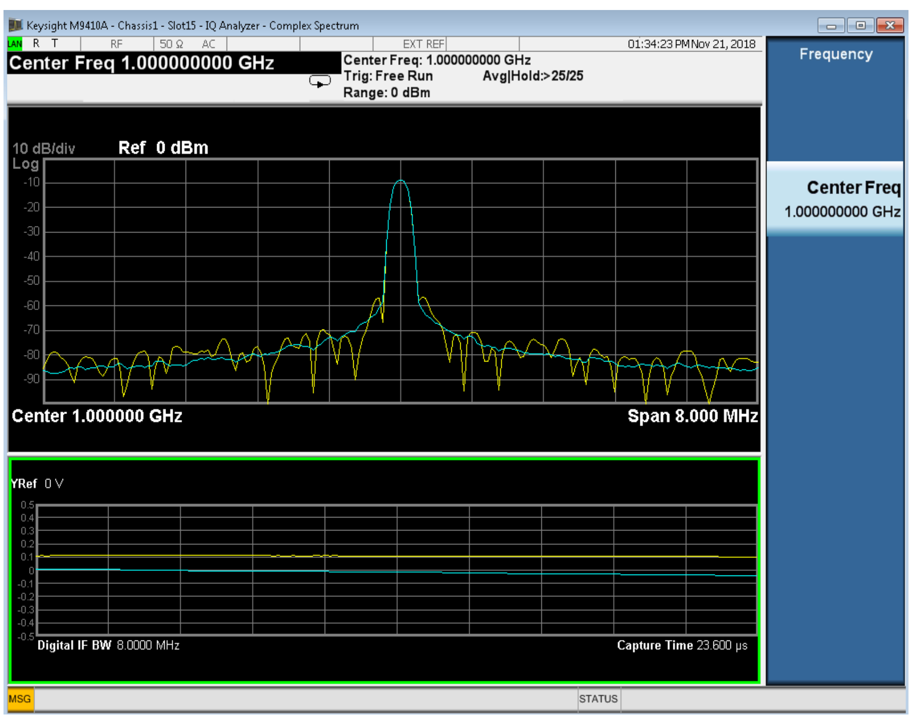
Set Receiver to Observe Signal

To set VXT receiver to observe the CW signal, please refer to the example code as below:

```
AgVisa32.viPrintf(xApp, ":INST:SEL BASIC;\n");//Enter basic mode (IQ Analyzer)

//It will take several seconds to load or switch mode, so this below code to
wait and check.
int tryTimes = 0; string queryResult;
do
{
AgVisa32.viPrintf(xApp, ":INST:SEL? \n");
AgVisa32.viRead(xApp, out queryResult, 1024);
tryTimes++;
}
while ((queryResult != "BASIC\n") && (tryTimes < 100));

AgVisa32.viPrintf(xApp, ":FREQ:CENT 1e9 Hz \n");//Set Frequency
AgVisa32.viPrintf(xApp, "INIT:CONT 1 \n"); //Set continuous sweep mode
AgVisa32.viPrintf(xApp, ":POW:RANG 10.0 \n");//Set Attenuator
```

Get the Measurement Result

Refer to the process of step 7 in example 1 to build and run your program to get the result as below.



It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

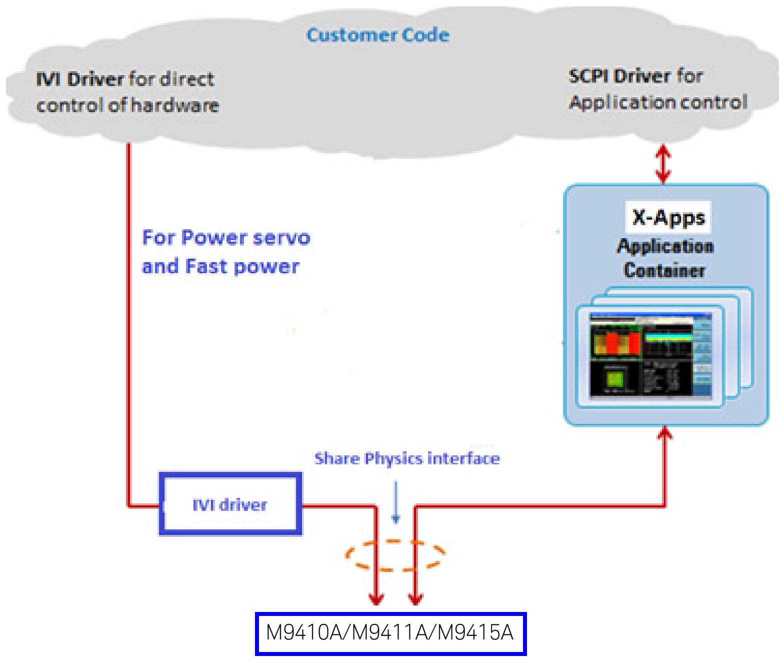
Basic Concepts: Two VXT Control Method

The VXT supports two method for remote control: by IVI driver and by SCPI command. The example 1 and 2 use IVI driver method and example 3 uses both methods.

IVI driver provides fast measurement speed and support power servo measurement with VXT.

SCPI command programming provides more features and Keysight classic UI display.

Figure 3-4 *IVI driver and SCPI driver in VXT*



The table below shows the supported measurement and features by IVI driver and SCPI.

Features	IVI Driver	SCPI
Spectrum Analysis	Supported	Supported
FFT Analysis	Supported	Supported
IQ Data Acquisition	Supported	Supported
Channel Power Test	Supported	Supported
Power Servo	Supported	Not supported
ACPR	Supported	Supported
Harmonics	Supported	Supported
OBW	Not supported	Supported
Spectrum Emission Mask	Not supported	Supported
Marker	Not supported	Supported

Features	IVI Driver	SCPI
Source (digital signal)*	Supported	Supported
Demod Digital Signal (EVM result)	Not supported	Supported
Keysight Classic Spectrum UI	Not supported	Supported

*. The VXT plays waveform file to produce digital signal.

SCPI Commands Control Method

There are two steps to use SCPI control method:

1. Setup VISA connection between PC and instrument
2. Send instrument SCPI command.

For example, `AgVisa32.viPrintf(xApp, ":FREQ:CENT 1e9 Hz\n");//Set receiver’s Center Freq` is a command used in example 3. `:FREQ:CENT 1e9 Hz` is a SCPI command.

To get a SCPI command, please refer to the online help system in VXT software or download the corresponding mode’s *User’s and Programmer’s Reference* from http://keysight.com/find/m9410a

For further information about SCPI command programming, please refer to *X-Series Programmer’s Guide*.

Example 4: Channel Power Acquisition

This example introduces the programming procedure to measure signal channel power with VXT.

- VXT source outputs a LTE FDD signal
- VXT receiver measures the signal channel power

NOTE N7624B Signal Studio for LTE/LTE-FDD is needed to play a LTE FDD signal with VXT product.

Before programming, please connect VXT RF Output to RF Input port and VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port. Please refer to Figure 3-1 for details.

The programming procedure are listed as 8 steps as below:

Step 1. - Create your project with Visual C#

Step 2. - Add References

Step 3. - Add Using Statements

Step 4. - Create and initialize the Instance

Step 5. - Write the Program

Step 6. - Close the Instance

Step 7. - Build and Run the Program

For step 1, 2, 3, 4, 6, 7, pleas refer to example 1 as those steps are similar. This section will only introduce the example code for step 5 - Write the Program.

After the VXT software is installed, you will find the source code in the directory below:

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsPowerAcquisition.

Write the Measurement Program

To output a LTE FDD signal with M941xA source, please refer to step 5 of example 2.

To measure the channel power of the LTE signal, please refer to the example code as below:

```
driver.Receiver.RF.Frequency = 1e9; //Set receiver center freq
driver.Receiver.RF.Power = -5;
driver.Receiver.RF.InputPort = Port.RFInput;

double RmsValue = driver.Source.Modulation.RmsPower;
double papr = -20 * Math.Log10(RmsValue);
driver.Receiver.RF.PeakToAverage = papr;
//Highlight: RF.PeakerToAverage is very important to receiver setting, when test
a modulated siganl. You can calculate this value based on Modulation.ArbRmsValue,
when you play a waveform on VXT.

driver.Apply(); //Apply the above setting to VXT receiver hardware

driver.Receiver.AcquisitionMode = AcquisitionMode.Power; //choose power
acquisition mode to test power.
driver.Receiver.PowerAcquisition.Bandwidth = 10e6;
//PowerAcquisition.Bandwidth set to a value > DUT signals's bandwidth.
driver.Receiver.PowerAcquisition.Duration = 0.02;
driver.Receiver.PowerAcquisition.ChannelFilter.Shape = ChannelFilterShape.None;
driver.Receiver.PowerAcquisition.ChannelFilter.Alpha = 0.1;
driver.Receiver.PowerAcquisition.ChannelFilter.Bandwidth = 6e6;
//ChannelFilter.Bandwidth set to the DUT signal's bandwidth.
driver.Apply(); //Apply the above setting to VXTII receiver's hardware.
driver.Receiver.Arm();//Start the M941xA's receiver's measurement

if (!driver.WaitForData((6000))) // It will take some time to change receiver
setting, so use WaitForData() method to wait for a while.
{
  throw new ApplicationException("WaitForData failed.  No acquisition was
made.");
}

 //Read the Channel Power's Measurement Result from VXTII memory, and Print it on
screen.
bool overloaded = false;
const int CAPTURE_ID = 0;
double power = 0;
driver.Receiver.PowerAcquisition.ReadPower(captureID: CAPTURE_ID, units:
PowerUnits.dBm, power: out power, overload: out overloaded);
```

Commands Summary

- `Arm()` method is used to trigger the data capture of acquisition. In this example, after all the hardware parameters are set, use `Arm()` to enable the power measurement. `driver.Receiver.Arm()` only triggers receiver's action, while `driver.Source.Arm()` only triggers source's action. `Driver.Arm()` triggers both the receiver and source's action together.
- After `Arm()` method to capture data, a time delay is set to wait for the measurement. `PowerAquisition.ReadPower()` is used to get the result from VXT memory.
- `Bandwidth, Duration, Offsetfreq` and `ChannelFilter` need to be set in power acquisition mode.
- `Receiver.RF` is used to set the basic RF parameters such as: `center freq`, `power (level)`, `input port`, and `peak to average`. In all the data acquisition modes, the commands to set those parameters are same. For example, use commands below to set the receiver’s RF parameters:

```
driver.Receiver.RF.Frequency = 1e9; //Set receiver’s center freq
driver.Receiver.RF.Power = -5; //Set receivers’ power range
driver.Receiver.RF.InputPort = Port.RFInput;
//Set the receiver’s RF input port
```

- In benchtop spectrum analyzer, the reference, attenuator and pre-amplifier is set to avoid mixer overload, or to control DANL. In VXT, only one parameter `RF.Power` is used to set the power range of a receiver. Set `RF.Power` value ≥ DUT signal to avoid mixer overload. In example 4, DUT LTE signal is –5 dBm, `RF.Power` is set to -5 dBm. VXT will set attenuator and pre-amp accordingly.
- `Receiver.RF.PeakToAverage` is very important to test a signal with high peak to average value, such as some cellular digital modulation signal. To test a -5 dBm LTE signal, the average channel power of this signal will be -5 dBm, set `Receiver.RF.Power` to -5 dBm. However this LTE signal could be 8 dB peak to average value, which means the peak signal power of this LTE signal should be -3 dBm. It exceeds -5 dBm `Receiver.RF.Power` setting, then the mixer will overload. The setting of `Receiver.RF.PeakToAverage` will help to optimize the attenuator and level setting in the mixer to get correct measurement result.

  In the source code below:

```
double RmsValue = driver.Source.Modulation.RmsPower;
double papr = -20 * Math.Log10(RmsValue);
driver.Receiver.RF.PeakToAverage = papr;
```

A waveform is played to generate this LTE signal, VXT provides a method to read an ARB RMS value form the waveform, it is a peak to average volt ratio value. It could be used to calculate power peak to average in dB unit.

Get the Measurement Result

Refer to the process of step 7 in example 1 to build and run your program to get the result as below.

```
Measure the Channel Power of a 5 MHz LTE Uplink Signal with Keysight VXTII M941x
A

Current SPI Config Value : 0x0Driver Initialized

VXTII M941xA Successfully Generated a 5 MHz LTE Uplink Signal.
Measured Channel Power : -5.11935086245377

If want to measure again, please press Enter button.
If Quit, please press stop.
_
```

Before running the program, please make sure the M9300A reference software is turned on.

It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

Basic Concepts: 4 Receiver Acquisition Mode

Keysight VXT M941xA provides 4 receiver acquisition mode as below.

Features	Use Case	Difference
Power Acquisition	Get the channel power directly	Fast power calculation
Spectrum Acquisition	Get the spectrum data based on span and RBW setting	More data points than FFT acquisition Lower DANL and better dynamic range
FFT Acquisition	FFT method to get the spectrum data	Faster SA data capture speed than spectrum acquisition Up to 512 data points Lower dynamic range
IQ Acquisition	IQ data output	Easy for post analysis

All these 4 receiver acquisition mode related methods are included in driver.Receiver menu.

Keysight VXT M941xA also provides 3 measurement mode: Power Servo, ACPR, Harmonics. Please refer example 8, 9, and 10 for details.

Example 5: Spectrum Acquisition

This example introduces the programming procedure to measure signal spectrum data and search the maximum power point with VXT.

- VXT source outputs a 1 GHz CW signal
- VXT receiver tests the signal spectrum data and search the maximum power point

Before programming, please connect VXT RF Output to RF Input port and VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port. Please refer to Figure 3-1 for details.

The programming procedure are listed as 8 steps as below:

Step 1. – Create your project with Visual C#

Step 2. – Add References

Step 3. – Add Using Statements

Step 4. – Create and initialize the Instance

Step 5. – Set VXT source to generate 1 GHz CW signal

Step 6. – Set VXT receiver to test signal spectrum data and search the maximum power point.

Step 7. – Close the Instance

Step 8. – Build and Run the Program

For step 1, 2, 3, 4, 5, 7, 8, pleas refer to example 1 as those steps are similar. This section will only introduce the example code for step 6 - Set VXT receiver.

After the VXT software is installed, you will find the source code in the directory below:

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsSpectrumAcquisition.

Set VXT Receiver to Test Spectrum Data

To set VXT receiver to test the signal spectrum data and search the maximum power point, please refer to the example code as below:

```
driver.Receiver.RF.Frequency = 1e9; //Set Receiver Center freq
driver.Receiver.RF.Power = -5; //Set power to -5 dBm
driver.Receiver.RF.InputPort = Port.RFInput;
driver.Apply(); //Apply the changes to hardware.

driver.Receiver.AcquisitionMode = AcquisitionMode.Spectrum; //Switch to Spectrum
Acquisition Mode.
driver.Receiver.SpectrumAcquisition.OffsetFrequency = 0;
driver.Receiver.SpectrumAcquisition.Span = 8e6; // Set Span of Spectrum
driver.Receiver.SpectrumAcquisition.ResolutionBandwidth = 30000; // Set RBW
driver.Receiver.SpectrumAcquisition.FFTWindowShape =
SpectrumFFTWindowShape.FlatTop; //Set FFT Window Shape
driver.Receiver.SpectrumAcquisition.Averaging.Mode = SpectrumAveraging.Time;
//Set Average's Mode - choose time based average.
driver.Receiver.SpectrumAcquisition.Averaging.Duration = 0.5; //Set average based
time.
driver.Receiver.SpectrumAcquisition.Averaging.Overlap = 0.5; //Set overlap value.

driver.Apply();
//Apply above receiver parameters' setting
driver.Receiver.Arm();
// Arm the digitizer to start measurement or data capture

const int CAPTURE_ID = 0;
double[] spectrum = new double[driver.Receiver.SpectrumAcquisition.Bins];
double fstart = 0;
double fdelta = 0; //Read the Spectrum data from VXTII's memory.
driver.Receiver.SpectrumAcquisition.ReadPowerSpectrum(CAPTURE_ID, PowerUnits.dBm,
ref spectrum, out overloaded, out fstart, out fdelta);

Search maximum data point. Source code:
int maxBin = FindMaximumAmplitude(ref spectrum);
private static int FindMaximumAmplitude(ref double[] vector)
{
     Double maxValue = Double.MinValue;
     int bin = -1;
     for (int i = 0; i < vector.Length; i++)
        {
     if (vector[i] > maxValue)
              {
               bin = i;
               maxValue = vector[i];
               }
         }
            return bin;
        }
```

Commands Summary

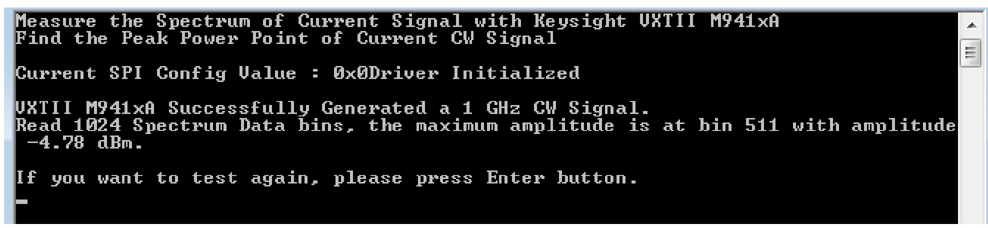
- `driver.Receiver.SpectrumAcquisition.Bins` is used to get the number of frequency points captured by the spectrum acquisition mode. When you set the Span and RBW value, the VXT will set the frequency bins value automatically. Increase span and decrease RBW will result in a larger Bin value, which means more frequency points.

  you program will assign enough space to save the spectrum data, based on Bin's value.

- `driver.Receiver.SpectrumAcquisition.ReadPowerSpectrum` is used to read spectrum acquisition. The default unit is dBm. The spectrum data captured by FFT acquisition mode is in mW unit.

Get the Measurement Result

Refer to the process of step 7 in example 1 to build and run your program to get the result as below.



Before running the program, please make sure the M9300A reference software is turned on.

It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

Example 6: FFT Acquisition

This example introduces the programming procedure to measure signal spectrum data and search the maximum power point with VXT.

- VXT source outputs a 1 GHz CW signal
- VXT receiver tests the signal spectrum data in FFT mode and search the maximum power point

Before programming, please connect VXT RF Output to RF Input port and VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port. Please refer to figure 3-1 for details.

The programming procedure are listed as 8 steps as below:

Step 1. – Create your project with Visual C#

Step 2. – Add References

Step 3. – Add Using Statements

Step 4. – Create and initialize the Instance

Step 5. – Set VXT source to generate 1 GHz CW signal

Step 6. – Set VXT receiver to test signal spectrum data in FFT mode and search the maximum power point.

Step 7. – Close the Instance

Step 8. – Build and Run the Program

For step 1, 2, 3, 4, 5, 7, 8, pleas refer to example 1 as those steps are similar. This section will only introduce the example code for step 6 - Set VXT receiver.

After the VXT software is installed, you will find the source code in the directory below:

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsFFTAcquisition.

Set VXT Receiver

To set VXT receiver to test the signal spectrum data in FFT mode and search the maximum power point, please refer to the example code as below:

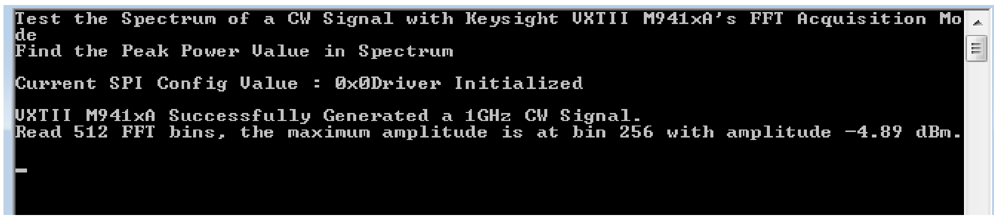
```
driver.Receiver.RF.Frequency = 1e9;
driver.Receiver.RF.Power = -5;
driver.Receiver.RF.InputPort = Port.RFInput;
driver.Receiver.RF.PeakToAverage = 3;
//PeakerToAverage is very important to receiver setting, when test a modulated
siganl. You can calculate this value based on Modulation.ArbRmsValue, if you play
a waveform on VXT. In this code, set the ReakToAverage value to 3 dB.

driver.Receiver.AcquisitionMode = AcquisitionMode.FFT;
driver.Receiver.FFTAcquisition.Length = FFTAcquisitionLength.Length_512;
driver.Receiver.FFTAcquisition.SampleRate = 5e6; //Sample rate should be set to a
value > 1.25 x Span.
driver.Receiver.FFTAcquisition.WindowShape = FFTWindowShape.FlatTop;
driver.Receiver.FFTAcquisition.Duration = 1e-4;
driver.Receiver.FFTAcquisition.ChannelFilter.Shape = ChannelFilterShape.None;
driver.Receiver.FFTAcquisition.ChannelFilter.Bandwidth = 4e6;
driver.Receiver.FFTAcquisition.ChannelFilter.Alpha = 0.1;
driver.Apply(); //Apply the changes to hardware.
driver.Receiver.Arm(); //Arm the digitizer

Double[] fftData = new Double[driver.FFTAcquisition.Samples];

driver.Receiver.FFTAcquisition.ReadMagnitudeData(0, ref fftData, out overloaded);

TodBm(ref fftData); //this method switch the FFT result into dBm value.
int maxBin = FindMaximumAmplitude(ref fftData); //This method find the peak power
value.

private static void TodBm(ref double[] vector)
   {
    for (int i = 0; i < vector.Length; i++)
       {
          vector[i] = 10 * Math.Log10(vector[i]);
        }
   }
```

Commands Summary

- `driver.Receiver.FFTAcquisition.Length` is limited up to 512 points to get fast test speed. To get more frequency points, please use spectrum acquisition mode.
- `driver.Receiver.FFTAcquisition.ReadMagnitudeData` is used to read spectrum acquisition. The default unit is dBm. The spectrum data captured by FFT acquisition mode is in mW unit.

Get the Measurement Result

Refer to the process of step 7 in example 1 to build and run your program to get the result as below.



Before running the program, please make sure the M9300A reference software is turned on.

It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

Example 7: IQ Acquisition

This example introduces the programming procedure to measure signal channel power with M941xA.

- VXT source outputs a LTE FDD signal
- VXT receiver captures signal's IQ data

NOTE The license key of N7624B Signal Studio is needed to play a LTE FDD signal with VXT product.

Before programming, please connect VXT RF Output to RF Input port and VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port. Please refer to figure 3-1 for details.

The programming procedure are listed as 8 steps as below:

Step 1. – Create your project with Visual C#

Step 2. – Add References

Step 3. – Add Using Statements

Step 4. – Create and initialize the Instance

Step 5. – Set VXT source to generate 1 GHz LTE FDD signal

Step 6. – Set VXT receiver to capture signal's IQ data.

Step 7. – Close the Instance

Step 8. – Build and Run the Program

For step 1, 2, 3, 4, 5, 7, 8, pleas refer to example 1 as those steps are similar. This section will only introduce the example code for step 6 – Capture IQ Data.

After the VXT software is installed, you can find the source code as below:

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsIQAcquisition.

Write the Measurement Program

To output a LTE FDD signal with M941xA source, please refer to Example 2.

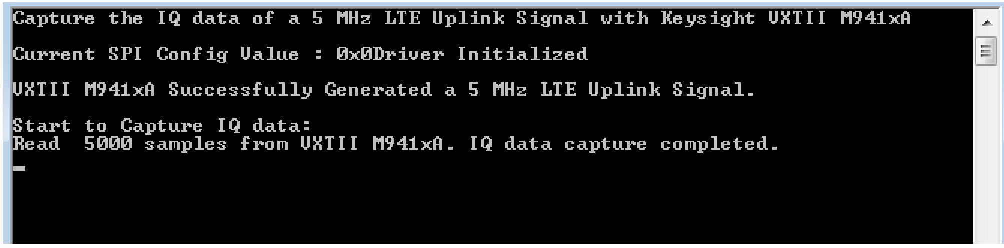
To capture the IQ data of the LTE signal, please refer to the example code as below:

```
driver.Receiver.RF.Frequency = 1e9; //Set the Receiver's Center Freq.
driver.Receiver.RF.Power = 0;
driver.Receiver.RF.InputPort = Port.RFInput; //set the Receiver to RF input
port.

double RmsValue = driver.Source.Modulation.RmsPower;
double papr = -20 * Math.Log10(RmsValue);
driver.Receiver.RF.PeakToAverage = papr;
// read the RMS value from waveform file and transfer it into Peak to Avearge
Ratio Value(PAR).

//Need to set a proper receiver RAR value. If the DUT signal's PAR is higher
than receiver, the receiver may overload!
driver.Receiver.AcquisitionMode = AcquisitionMode.IQ;
//choose data acquirement mode to IQ Acquisition Mode
driver.Receiver.IQAcquisition.SampleRate = 5000000; //Set Sample Rate
double DURATION = 1e-3;
double SAMPLE_RATE = driver.Receiver.IQAcquisition.SampleRate;
driver.Receiver.IQAcquisition.Samples = (int)(DURATION * SAMPLE_RATE);
//Set the IQ data acquisition sample number.
int samples = driver.Receiver.IQAcquisition.Samples;
driver.Receiver.IQAcquisition.ChannelFilter.Shape = ChannelFilterShape.None;
//Set the IQ Acquisition's channel filter shpe

driver.Apply(); //Apply the changes to hardware.

driver.Receiver.Arm();
//Arm the digitizer to start measurement or data capture
//After Arm() method, the IQ data will be captured into VXTII's memory, and we
will use IQAcquisition.ReadIQData() method to read the data.

double[] interleavedIqBlock = null; // Allocate enough room for 5,000 samples

Console.WriteLine("Start to Capture IQ data:");
driver.Receiver.IQAcquisition.ReadIQData(0, IQUnits.SquareRootMilliWatts, 0,
samples, ref interleavedIqBlock, out overloaded);
//Read the captured IQ data from VXT.

//Add code to process this IQ data per your own requirement.
Console.WriteLine("Read {0,5} samples from VXT M941xA. IQ data capture
completed.", samples);
```

- `driver.Receiver.IQAcquisition.Samples` is used to set the sample points you want to capture, so you can defined a number directly. Usually we can define it according to `IQAcquisition.SampleRate` and Duration Time you want to test. So in source code, it set as below,
  `driver.Receiver.IQAcquisition.Samples = (int)(DURATION * SAMPLE_RATE);`
  //Set the IQ data acquisition sample number.

Get the Measurement Result

Refer to the process of step 7 in example 1 to build and run your program to get the result as below.



Before running the program, please make sure the M9300A reference software is turned on.

It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

Example 8: Power Servo Measurement

This example introduces the programming procedure to perform power servo, and DUT is only a cable.

Power Servo Loop

One of the key measurements for a power amplifier or chip set, is performing a Servo Loop. Because when you measure a power amplifier or chip set, it is typically specified at a specific output power level. It needs to adjust the source input level until you measure the exact power level. To do this, you will continually adjust the source until you achieve the specified output power level.

NOTE In this example, need to generate a WCDMA uplink signal with VXT's source, so the license key of N7600B Signal Studio is needed with VXT product.

Before programming, please connect VXT RF Output to RF Input port and VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port. Please refer to figure 3-1 for details.

The programming procedure are listed as 8 steps as below:

Step 1. – Create your project with Visual C#

Step 2. – Add References

Step 3. – Add Using Statements

Step 4. – Create and initialize the Instance

Step 5. – Set VXT source to generate 1 GHz WCDMA Uplink signal

Step 6. – Perform Power Servo operation.

Step 7. – Close the Instance

Step 8. – Build and Run the Program

For step 1, 2, 3, 4, 5, 7, 8, pleas refer to example 1 as those steps are similar. This section will only introduce the example code for step of Write the Program.

After the VXT software is installed, you can find the source code as below:

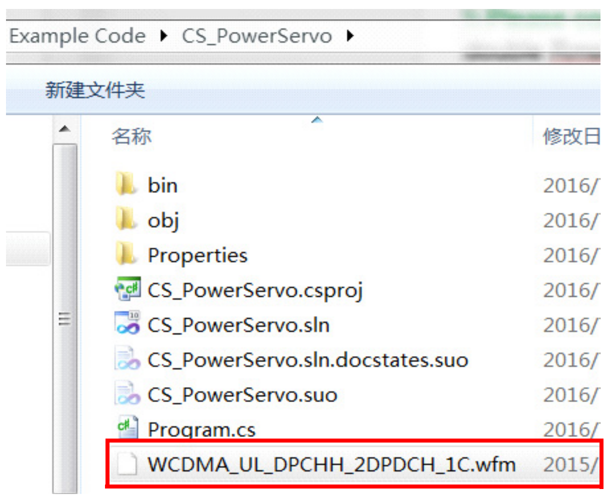
C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsMeasurement_PowerServo.

Write the Measurement Program

To output a WCDMA uplink signal with the M941xA source, please refer to below source codes:

```
driver.Source.RF.Frequency = 1e9;
driver.Source.RF.Level = -20;
//Set source's original output power level to perform power servo.
driver.Source.RF.OutputPort = Port.RFOutput;
driver.Source.RF.OutputEnabled = true;
driver.Source.LoadWaveform("..\\..\\..\\", "WCDMA_UL_DPCHH_2DPDCH_1C.wfm");
//Because the waveform file WCDMA_UL_DPCHH_2DPDCH_1C.wfm, is put in the root folder of
current project, the LoadWaveform's address is "..\\..\\..\\".
driver.Source.Modulation.PlayArb("WCDMA_UL_DPCHH_2DPDCH_1C.wfm",
StartEvent.Immediate);
driver.Source.Modulation.Enabled = true;
driver.Apply();
```

The waveform file “*WCDMA_UL_DPCHH_2DPDCH_1C.wfm*” used in this example is attached in the project file. You can set it to other address in `Source.LoadWaveform()`.



To perform power servo operation, please refer to the example code as below:

```
// Setup Receiver
driver.Receiver.RF.Frequency = 1e9;
driver.Receiver.RF.Power = 5;
double RmsValue = driver.Source.Modulation.RmsPower;
double papr = -20 * Math.Log10(RmsValue);
driver.Receiver.RF.PeakToAverage = papr;
//Set the Peak to Average value according to RMS power of waveform loaded in the
source's modulator
driver.Receiver.RF.InputPort = Port.RFInput;
driver.Apply();

//Configure Power Servo
driver.Measurement.PowerServo.AcquisitionMode = AcquisitionMode.FFT;
driver.Receiver.FFTAcquisition.SampleRate = 30.72e6;
driver.Receiver.FFTAcquisition.Length = FFTAcquisitionLength.Length_512;
driver.Receiver.FFTAcquisition.Duration = 0.0001;
driver.Receiver.FFTAcquisition.WindowShape = FFTWindowShape.Gaussian;

ChannelFilterShape FilterType = ChannelFilterShape.RaisedCosine;
double FilterAlpha = 0.22;
double FilterBw = 3840000.0;
driver.Receiver.FFTAcquisition.ChannelFilter.Configure(FilterType, FilterAlpha,
FilterBw);

double Level = -20; double Gain = 5;
driver.Measurement.PowerServo.InputPower = Level + Gain;
driver.Measurement.PowerServo.OutputPower = Level;
driver.Measurement.PowerServo.OutputPowerMargin = 0.05;
driver.Measurement.PowerServo.OverheadTime = 600e-6;
driver.Measurement.PowerServo.MaximumOutputPower = 20;

driver.Source.Apply();//Apply source's setting to VXT's source
driver.Receiver.Apply();/Ch/Apply receiver's setting to VXT's receiver

Measurements[] measlist = new Measurements[] { Measurements.PowerServo };
driver.Measurement.SetEnableList(measlist);
//choose measurement mode, Power Servo in this case, then load Measurement related
parameters to VXT receiver's hardware.

driver.Measurement.Process();//Active the measurement
double MeasuredPower = 0;
bool ServoPass = false;
int ServoCount = 0;
bool Overload = true;

//Read Power Servo Result
driver.Measurement.PowerServo.ReadPowerServo(out MeasuredPower, out ServoPass, out
Overload, out ServoCount);
```

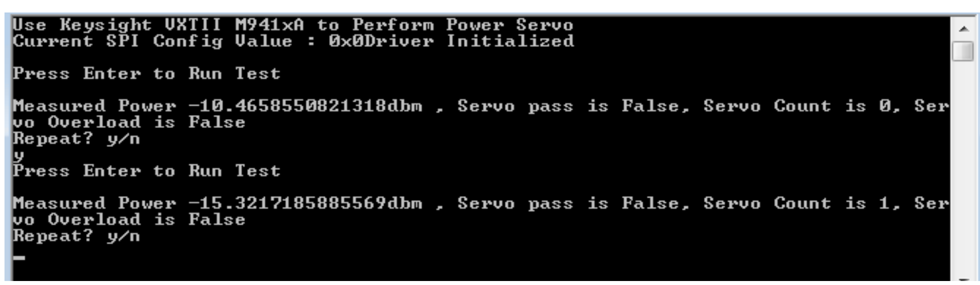
Commands Summary

- In this power servo example, the `driver.Arm()` method is not used to active the measurement. We used the `driver.measurement.process()` method.
- The `driver.Arm()` method is mainly used for the 4 basic acquisition modes, Power acquisition, FFT acquisition, spectrum acquisition and IQ acquisition. It will enable the test without any time delay or waiting.
- The `driver.measurement.process()` is mainly used to active the 3 measurement function, Power Servo, Harmonics and ACPR. A basic data acquisition operation only perform one time data acquisition, so we use `driver.Arm()`. The 3 measurements function will perform several times data acquiring to output a result. Take Harmonics for example, it tests the main signal and harmonics signal amplitude one by one. So the measurement will `use measurement.process()` method.
- The `driver.measurement.SetEnableList()`is used to choose measure mode, Power Servo, Harmonics or ACPR, and load the measurement related parameters to VXT receiver's hardware. `Driver.Apply()` only apply or load source and receiver's parameters, so it will not load Measurement related parameters. In this example case, `driver.Receiver.Apply()`, `driver.Measurement.SetEnableList()`, and `driver.Measurement.Process()` are used after setting all the receiver and measurement parameters.
- The 3 measurements items are tested based on 1 of 4 basic acquisition modes, so it requires to choose the `Measurement.PowerServo.AcqusitionMode`, and set related parameters in this mode.

Get the Measurement Result

Refer to the process of step 7 in example 1 to build and run your program to get the result as below.

The measured power is –14.98 dBm, and the servo count is 4.



Before running the program, please make sure the M9300A reference software is turned on.

It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

Example 9: Harmonics Measurement

This example introduces the programming procedure to measure harmonics of a CW signal with M941xA.

- VXT source outputs a 1 GHz CW signal
- VXT receiver test harmonics of this signal

Before programming, please connect VXT RF Output to RF Input port and VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port. Please refer to figure 3-1 for details.

The programming procedure are listed as 8 steps as below:

Step 1. – Create your project with Visual C#

Step 2. – Add References

Step 3. – Add Using Statements

Step 4. – Create and initialize the Instance

Step 5. – Set VXT source to generate 1 GHz CW signal

Step 6. – Set VXT receiver to test the harmonics.

Step 7. – Close the Instance

Step 8. – Build and Run the Program

For step 1, 2, 3, 4, 5, 7, 8, pleas refer to example 1 as those steps are similar. This section will only introduce the example code for step 6 – Set VXT receiver to test harmonics.

After the VXT software is installed, you can find the source code as below:

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsMeasurement_Harmonics.

Write the Measurement Program

To output a CW signal with VXT source, please refer to Example 1.

To test the harmonics of this signal, please refer to below source codes:

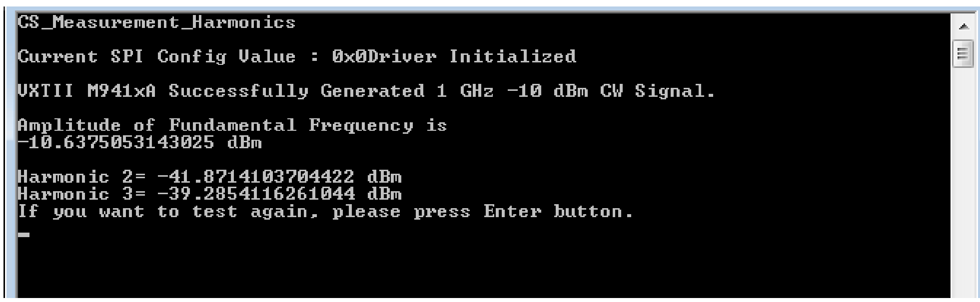
```
driver.Receiver.RF.Frequency = 1e9;
driver.Receiver.RF.Power = 10;
//Set the Receiver.RF.Power a little bigger than target test signal to avoid
overload
valuedriver.Receiver.RF.PeakerToAverage =3;
driver.Receiver.RF.InputPort = Port.RFInput;

//Use spectrum mode to test harmonics and set spectrum parameters
driver.Measurement.Harmonics.AcquisitionMode = AcquisitionMode.Spectrum;
driver.Receiver.SpectrumAcquisition.Span = 1e6;
driver.Receiver.SpectrumAcquisition.ResolutionBandwidth = 1e3;
driver.Receiver.SpectrumAcquisition.FFTWindowShape =
SpectrumFFTWindowShape.Hann;
driver.Apply();// Apply the above setting to VXTII receiver's hardware.

driver.Measurement.Harmonics.Configure(fundamentalFrequency:
driver.Source.RF.Frequency,maximumHarmonicsNumber: 3);
//Configure Harmonic measurement parameters, the fundament frequency and max
harmonics numbers.

Measurements[] measlist = new Measurements[] { Measurements.Harmonics };
driver.Measurement.SetEnableList(measlist);
//Choose the measurement mode, then load related parameters to VXT receiver's
hardware.

driver.Measurement.Process();
//Active the harmonics measurement.

bool overload = false;
double[] harmData = new double[5];
bool[] overloads = new bool[5];

driver.Measurement.Harmonics.ReadHarmonics(Harmonics: ref harmData, Overload:
ref overload);
//Read Harmonics test result. The output is the power of main signal and
harmonics in dBm unit.
```

Commands Summary

- Maximum Harmonics Number is to set the number of harmonics will be tested, default vaule is 1. Please take note it includes the fundamental frequency signal. For example, if you set the `driver.Measurement.Harmonics.MaxmumHarmonicsNumber` to 3 to test a 1 GHz CW signal, it will test the power level of 1 GHz(fundamental freq), 2 GHz and 3 GHz signal.
- `Measurement.Harmonics.ReadHarmonics()` is used to read the Harmonics test result, and the result is only in dBm unit. If you want to get the result in dBc unit, such as 2nd Harmonics is XX dBc related to fundamental signal, you need to calculate with your own code
- Because the Harmonics test usually requires high dynamic range, please carefully adjust `driver.receiver.RF.power` to achieve a better result.
- IQ Acquisition mode is not supported by Harmonics measurement mode.

Get the Measurement Result

Refer to the process of step 8 in example 1 to build and run your program to get the result as below.



```
CS_Measurement_Harmonics

Current SPI Config Value : 0x0Driver Initialized

VXTII M941xA Successfully Generated 1 GHz -10 dBm CW Signal.

Amplitude of Fundamental Frequency is
-10.6375053143025 dBm

Harmonic 2= -41.8714103704422 dBm
Harmonic 3= -39.2854116261044 dBm
If you want to test again, please press Enter button.
_
```

Before running the program, please make sure the M9300A reference software is turned on.

It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

Example 10: ACPR Test

This example introduces the programming procedure to measure ACPR of a LTE FDD signal with M941xA.

- VXT source outputs a LTE FDD signal
- VXT receiver test ACPR of this signal

Before programming, please connect VXT RF Output to RF Input port and VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port. Please refer to figure 3-1 for details.

The programming procedure are listed as 8 steps as below:

Step 1. - Create your project with Visual C#

Step 2. - Add References

Step 3. - Add Using Statements

Step 4. - Create and initialize the Instance

Step 5. - Write the Program (Set VXT source to generate LTE FDD signal, and set VXT receiver to test the ACPR of this signal)

Step 6. - Close the Instance

Step 7. - Build and Run the Program

For step 1, 2, 3, 4, 6, 7, please refer to example 1 as those steps are similar. This section will only introduce the example code for step 5 - Write the Program.

After the VXT software is installed, you can find the source code as below:

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsMeasurement_ACPR.

Write the Measurement Program

To output a CW signal with VXT source, please refer to Example 2.

To test the harmonics of this signal, please refer to below source codes:

```
//Set Receiver
driver.Receiver.RF.Frequency = 1e9;
driver.Receiver.RF.Power = -10;
//the Receiver.RF.Power should be set equal to the target test value, to get
exact test result.
double RmsValue = driver.Source.Modulation.RmsPower;
double papr = -20 * Math.Log10(RmsValue);
// read the RMS value from waveform file and transfer it into Peak to Avearge
Ratio Value(PAR).
driver.Receiver.RF.PeakToAverage = papr;
//Need to set a proper receiver RAR value.
//If the DUT signal's PAR is higher than receiver, the receiver may overload!!
driver.Apply();

//User Power mode
driver.Measurement.Acpr.AcquisitionMode = AcquisitionMode.FFT;
driver.Measurement.Acpr.UseChanPwrForRef = false;
//VXTII could use power servo channel power result as the carrier channel power
to test ACPR.
//This will help to improve the test speed if customer want to perform
powerservo and then test ACPR.
//This example will only show how to test ACPR, so the UseChanPwrForRef set to
false.

int numAcprMeas = 3;
//In this code, we will test 2 adjcent channels(1 lower and 1 upper), so 3
channel powers will be tested to get 2 ACPR results.
double[] AcprOffsetFreq = new double[] { 0, -5e6, 5e6 };
double[] AcprSpan = new double[] { 4.5e6, 4.5e6, 4.5e6 };
double[] AcprDuration = new double[] { 500e-6, 500e-6, 500e-6 };
ChannelFilterShape[] AcprFilterType = new ChannelFilterShape[numAcprMeas];
AcprFilterType[0] = ChannelFilterShape.Rectangular;
AcprFilterType[1] = ChannelFilterShape.Rectangular;
AcprFilterType[2] = ChannelFilterShape.Rectangular;
double[] AcprAlpha = new double[] { 0.22, 0.22, 0.22 };
double[] AcprBandWidth = new double[] { 3.84e6, 3.84e6, 3.84e6 };

Measurements[] measlist= new Measurements[]{Measurements.Acpr};

driver.Measurement.Acpr.SetAcprParameter(AcprOffsetFreq, AcprSpan,
AcprDuration);
driver.Measurement.Acpr.AveragingNumber = 1;
```

```
driver.Measurement.Acpr.ConfigureFilter(AcprFilterType, AcprAlpha, AcprBandWidth);
driver.Measurement.SetEnableList(measlist);
//Choose the measurement mode, then load related parameters to VXT receiver's
hardware.

driver.Measurement.Process();
//Active the harmonics measurement.
double[] AcprResultPower = new double[numAcprMeas];
bool[] AcprResultOverload = new bool[numAcprMeas];
// Read acpr result
driver.Measurement.Acpr.ReadAcpr(ref AcprResultPower, ref AcprResultOverload)

// Print Results
Console.WriteLine("Main Channel Power: " + driver.Measurement.Acpr.ReferencePower +
" dBm/4.5 MHz");
//Print reference signal power
Console.WriteLine("Offset Freq 5 MHz, Lower:"+AcprResultPower[1]+"dBC/4.5 MHz");
Console.WriteLine("Offset Freq 5 MHz, Lower:"+AcprResultPower[2]+"dBC/4.5 MHz");
//Output 2 adjacent channel powers
```

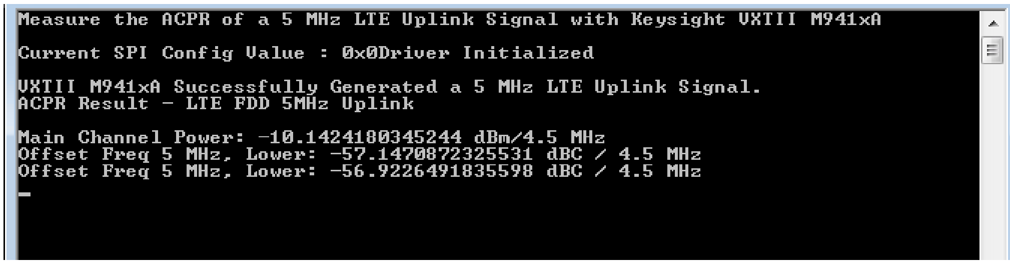
Commands Summary

- `driver.Measurement.Acpr.UseChanPwrForRef` is a special setting for VXT's ACPR measurement. If it set to `true`, VXT will use Power servo's power result as carrier channel's power to perform ACPR test. Because VXT is designed to perform high speed measurement. When customer test a amplifier, it requires to perform power servo and ACPR for same DUT. VXT support to use power servo's power result as carrier channel's power to perform ACPR, so it will help to decrease the total measurement time.
- If you don't need to perform Power Servo (such as transmitter or base station test) before ACPR test, should set the `driver.Measurement.Acpr.UseChanPwrForRef` to `false`. Current example is this case.
- `Measurement.Acpr.ReadAcpr` will get a `AcprResultOverload` value, and it will help to check whether you have received correct result. If the AcprResultOverload get **true,** the ACPR power result will be incorrect as the VXT has already overloaded.

Get the Measurement Result

Refer to the process of step 7 in example 1 to build and run your program to get the result as below.

VXT test the ACPR result as below:



Before running the program, please make sure the M9300A reference software is turned on.

It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

Example 11: Combined WCDMA Power Servo and ACPR Measurement

In Power Servo and ACPR measurement, Servo is performed by "Baseband Tuning" to adjust the source amplitude and then "Baseband Tuning" is used to digitally tune the center frequency in order to make channel power measurements, at multiple offsets, using the Power Servo interface of the VXT.

The following example code demonstrates how to instantiate driver instances, set the resource names and various initialization values, initialize the driver instances, and perform the other relevant tasks:

- Send source RF and LoadWaveform commands to the VXT driver
- Send receiver RF commands to the VXT driver
- Send measurement process command to run a Servo loop and ACPR measurement
- Read the measurement result and close the driver

Before programming, please connect VXT RF Output to RF Input port and VXT 100 MHz Ref In port to M9300A's 100 MHz Ref Out port. Please refer to figure 3-1 for details.

After the VXT software is installed, you can find the source code as below:

C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM941x x.x.x\Examples\CSharp\CsPowerServo_ACPR.

Example Program - Pseudo - code

Initialize drivers for VXT and check for errors

Configure Source RF Settings:

Frequency

RF Level

RF Output Port and Enable On

Configure ARBPLAY Settings:

Load WCDMA Signal Studio File

Get RMS Value

Play ARB File

Configure Receiver RF Settings:

Frequency

Level

Peak to Average Ratio

Input Port

- Configure Power Servo Settings
  - Enable Power Servo Measurement
  - Acquisition Mode
  - Acquisition Settings
  - Power Servo Settings
- Configure ACPR Settings
  - Enable ACPR Measurement
  - ACPR Measurement Settings
- Enable VXT Settings:
  - Source Settings
  - Receiver Settings
- Apply All Above Settings and Measurements
- Read Power Servo Results
  - Measured Power
  - Pass/Fail
  - Overload
  - Servo Count
- Read ACPR Results
  - ACPR Values
  - Overload

Source Code

```
// Copy the following example code and compile it as a C# Console Application
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Keysight.KtM941x;
#endregion

namespace CS_PowerServo_ACPR
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create driver instances
            Console.WriteLine("Perform Power Servo, then Measure ACPR");
            KtM941x driver = null;
            try
            {
                #region Initialize Driver Instances

                string ResourceName = "PXI21::0::0::INSTR";
                bool IdQuery = true;
                bool Reset = true;
                string OptionString = "QueryInstrStatus=true, Simulate=false,
DriverSetup= Model=M941xAe";
                driver = new KtM941x(ResourceName, IdQuery, Reset, OptionString);
                Console.WriteLine("Driver Initialized\n");
                #endregion

                #region Check Instrument Queue for Errors

                int errorcode = 0;
                string message = string.Empty;
                // Check instrument for errors
                do
                {
                    Ivi.Driver.ErrorQueryResult err = driver.Utility.ErrorQuery();
                    message = err.Message;
                    if (errorcode != 0)
                    {
                        Console.WriteLine(message);
                    }
                } while (errorcode != 0);

                #endregion
                #region Create Default Settings for WCDMA Uplink Signal
                // Source Settings
                double Frequency = 1000000000.0;
                double Level = -20;
                double Gain = 5;
                double PowerOutMargin = 0.05;
                double ServoOverheadTime = 600e-6;
               //Software License is needed if a Signal Studio waveform file is used.
               string WaveformFile = "WCDMA_UL_DPCHH_2DPDCH_1C.wfm";
              string ExamplesFolder = "..\\..\\..\\"; // Because we put this waveform
in the sample code project's root folder, we use this address.
```

```
                // Receiver Settings
                double ChannelTime = 0.0001;
                double AdjacentTime = 0.0005;
                double IfBandwidth = 40000000.0;
                double MeasureBW = 5000000.0;
                ChannelFilterShape FilterType = ChannelFilterShape.RaisedCosine;
                double FilterAlpha = 0.22;
                double FilterBw = 3840000.0;
                double AcprFliterBw = 3840000.0;
                double AcprFilterAlpha = 0.22;
                ChannelFilterShape AcprFilterType =
ChannelFilterShape.RaisedCosine;
                double[] FreqOffset = new double[] {-5000000.0, 5000000.0,
-10000000.0, 10000000.0};
                double[] acprFilterAlpha = new double[4] {AcprFilterAlpha,
AcprFilterAlpha, AcprFilterAlpha, AcprFilterAlpha};
                double[] acprFilterBw = new double[4] {AcprFliterBw,
AcprFliterBw, AcprFliterBw, AcprFliterBw};
                ChannelFilterShape[] acprFilterType = new ChannelFilterShape[]
{AcprFilterType, AcprFilterType, AcprFilterType, AcprFilterType};
                double AcprSpan = 30.72e6 / 1.25;
                double AcprDuration = AdjacentTime;
                double[] acprSpan = new double[4] {AcprSpan, AcprSpan, AcprSpan,
AcprSpan};
                double[] acprDuration = new double[4] {AcprDuration,
AcprDuration, AcprDuration, AcprDuration};
                double MeasuredPower = 0;
                bool ServoPass = false;
                int ServoCount = 0;
                bool Overload = true;
                double[] MeasuredACPR = new double[5];
                bool[] MeasuredACPROverload = new bool[5];
                double RmsValue = 0;
                #endregion

                #region Run Commands
                //Setup Source
                driver.Source.LoadWaveform(ExamplesFolder, WaveformFile);
                RmsValue = driver.Source.Modulation.RmsPower;
                double papr = -20 * Math.Log10(RmsValue);
                driver.Source.RF.Frequency = Frequency;
                driver.Source.RF.Level = Level;
                driver.Source.RF.OutputPort = Port.RFOutput;
                driver.Source.RF.OutputEnabled = true;
                driver.Source.Modulation.PlayArb("WCDMA_UL_DPCHH_2DPDCH_1C.wfm",
StartEvent.Immediate);
```

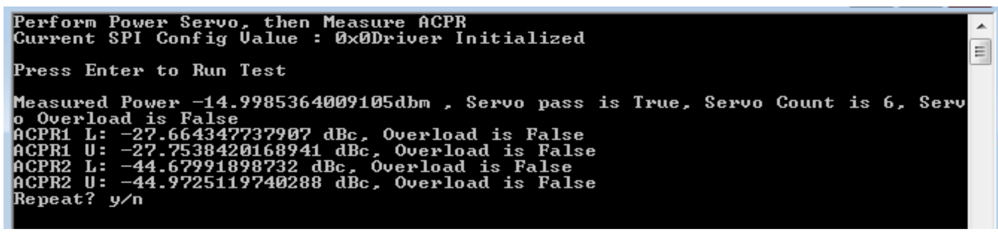
```
                driver.Source.Modulation.Enabled = true;
                driver.Apply();

                // Setup Receiver
                driver.Receiver.RF.Frequency = Frequency;
                driver.Receiver.RF.Power = Level + Gain;
                driver.Receiver.RF.PeakToAverage = papr;
                driver.Receiver.RF.InputPort = Port.RFInput;

                // Configure PowerServo
                driver.Measurement.PowerServo.AcquisitionMode = AcquisitionMode.FFT;
                driver.Receiver.FFTAcquisition.SampleRate = 30.72e6;
                driver.Receiver.FFTAcquisition.Length =
FFTAcquisitionLength.Length_512;
                driver.Receiver.FFTAcquisition.Duration = ChannelTime;
                driver.Receiver.FFTAcquisition.WindowShape =
FFTWindowShape.Gaussian;
                driver.Receiver.FFTAcquisition.ChannelFilter.Configure(FilterType,
FilterAlpha, FilterBw);
                driver.Measurement.PowerServo.InputPower = Level + Gain;
                driver.Measurement.PowerServo.OutputPower = Level;
                driver.Measurement.PowerServo.OutputPowerMargin = PowerOutMargin;
                driver.Measurement.PowerServo.OverheadTime = ServoOverheadTime;
                driver.Measurement.PowerServo.MaximumOutputPower = 20;

                //Configure Acpr
                driver.Measurement.Acpr.AcquisitionMode = AcquisitionMode.FFT;
                driver.Measurement.Acpr.UseChanPwrForRef = true;
                driver.Measurement.Acpr.ConfigureFilter(acprFilterType,
acprFilterAlpha, acprFilterBw);
                driver.Measurement.Acpr.SetAcprParameter(FreqOffset, acprSpan,
acprDuration);

                Measurements[] measlist = new Measurements[]
{Measurements.PowerServo, Measurements.Acpr};
                driver.Measurement.SetEnableList(measlist);

                string response = "y";
                while (string.Compare(response, "y") == 0)
                {
                    Console.WriteLine("Press Enter to Run Test");
                    Console.ReadLine();
                    //Process measurement
                    driver.Measurement.Process();
                    // Check instrument for errors
                    do
```

```
                    {
                        Ivi.Driver.ErrorQueryResult err =
driver.Utility.ErrorQuery();
                        message = err.Message;
                        if (errorcode != 0)
                        {
                            Console.WriteLine(message);
                        }
                    } while (errorcode != 0);

                    //Read PowerServo
                    driver.Measurement.PowerServo.ReadPowerServo(out
MeasuredPower, out ServoPass, out Overload, out ServoCount);
                  Console.WriteLine("Measured Power {0}dbm , Servo pass is {1},
Servo Count is {2}, Servo Overload is {3}", MeasuredPower, ServoPass,
ServoCount, Overload);
                    driver.Measurement.Acpr.ReadAcpr(ref MeasuredACPR, ref
MeasuredACPROverload);
                    Console.WriteLine("ACPR1 L: {0} dBc, Overload is {1}",
MeasuredACPR[0], MeasuredACPROverload[0]);
                    Console.WriteLine("ACPR1 U: {0} dBc, Overload is {1}",
MeasuredACPR[1], MeasuredACPROverload[1]);
                    Console.WriteLine("ACPR2 L: {0} dBc, Overload is {1}",
MeasuredACPR[2], MeasuredACPROverload[2]);
                    Console.WriteLine("ACPR2 U: {0} dBc, Overload is {1}",
MeasuredACPR[3], MeasuredACPROverload[3]);
                    Console.WriteLine("Repeat? y/n");
                    response = Console.ReadLine();
                }
                #endregion
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exceptions for the drivers:\n");
                Console.WriteLine(ex.Message);
            }
            finally
            {
                if (driver != null)
                {
                    // Close the driver
                    driver.Close();
                    Console.WriteLine("");
                    Console.WriteLine("Driver Closed");
                }
            }
```

```
                Console.WriteLine("Done - Press Enter to Exit");
                Console.ReadLine();

            }
        }
    }
```

Get the Measurement Result

Refer to the process of step 7 in example 1 to build and run your program to get the result as below.



Before running the program, please make sure the M9300A reference software is turned on.

It will take several minutes to run the program as the VXT vector transceiver need boot up before running this program.

KEYSIGHT TECHNOLOGIES

Edition 1, July 2021

M9410-90009

www.keysight.com