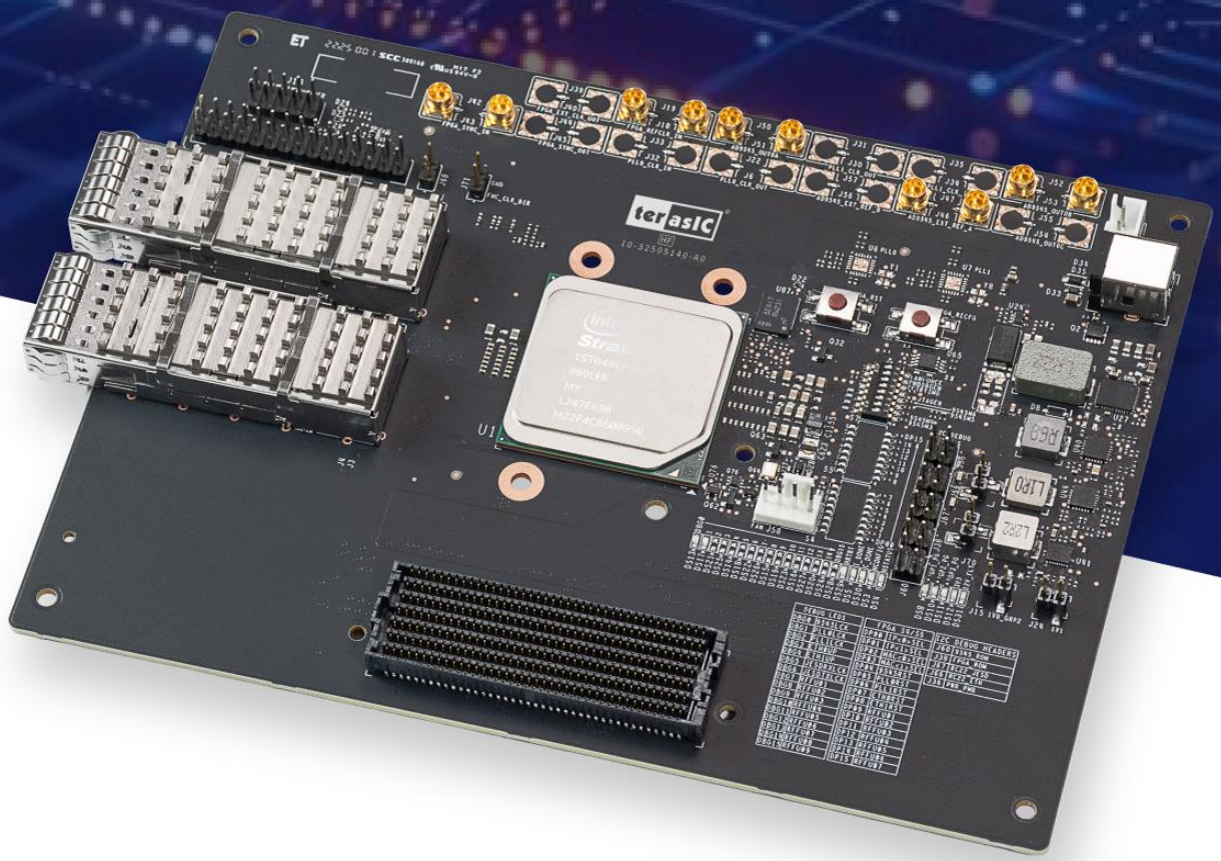
Software Setup and Installation Guide Document



HSB 100G User Guide

© Copyright Terasic Inc. All Rights Reserved

CONTENTS

Chapter 1

Introduction

This guide provides instructions for the installation, configuration, and use of the Stratix 10 Sensor Processing Kit in 100G configuration and especially the demonstration application and software components that come with it. The kit will hereafter be referred to simply as **HSB** throughout this document.

In the provided application, the Signal Generator, a waveform generator using simple algebraic and trigonometric functions provides a digital signal (using "in-phase" and "quadrature" components – ie I/Q samples) that is transmitted via RoCE protocol to the HSB board's FPGA and over JESD to an attached ADI board (AD9986 MxFE ADC/DAC transmitter/receiver). The ADI board DACs convert this signal to an analog RF signal at a rate of 983.04 Msps. The RF signal is sent to an oscilloscope via DAC0 and is looped back from DAC1 into the ADI board's ADC0. The AD9986 sends ADC data over the JESD lanes to the FPGA, and the FPGA sends ADC0 data over the QSFP via RoCE and into the host system running the signal generator application. The Host displays this received data from ADC0 in both the time domain and frequency domain. A diagram showing the hardware layout and data flow is shown below.

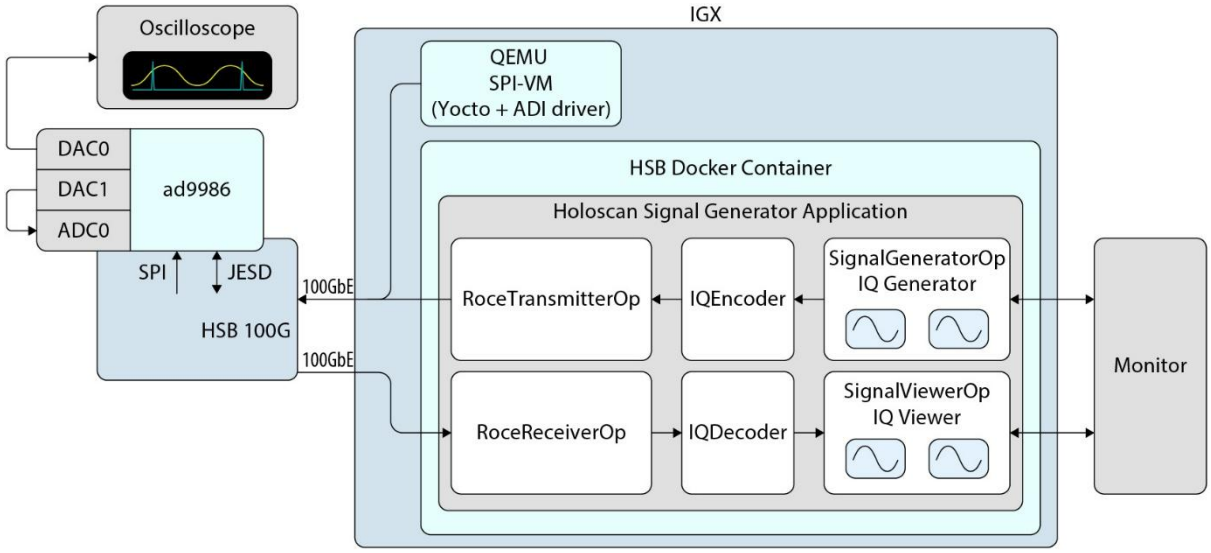


Figure 1-1 Signal Generator application and hardware layout of NVIDIA IGX Dev Kit + HSB 100G + AD9986

The operators and applications working within the Holoscan Sensor Bridge and Holoscan SDK software components are provided and described "as-is" to facilitate developer onboarding.

Additionally, to facilitate compatibility of the Holoscan Sensor Bridge framework with the ADI board configuration, a VM-based device configuration is provided. This provides for user-space application and driver development for the ADI boards within the typical host system (IGX Orin) and HSB framework.

This configuration utilizes a Yocto-based VM, kernel drivers from Analog Devices, and a driver that captures SPI commands from the kernel drivers. The SPI commands are forwarded to the ADI devices (AD9986 and HMC7044) over the FMC connector via the HSB's SPI interfaces.

1.1 AD9986 and JESD Configuration

☐ The AD9986 configuration is based on the following use case.

Description	Clock (GHz)		JESD204x Mode[1]		Tx Interpolation		Rx Decimation		Tx Data Rate (MSPS)	No. of Tx Channels	Rx Data Rate (MSPS)	No. of Rx Channels	Link Line Rate (Gbps/lane)	JESD204x Protocol[2]
	Tx	Rx	Tx	Rx	Coarse	Fine	Coarse	Fine						
mmWave 5G, 4T4R single band, 983.04 MSPS I/Q	11.79648	2.94912	15C	16C	12	1	3	1	983.04	4	983.04	4	16.22016	C

Figure 1-2 The AD9986 configuration

Note that the above configures the ADC/DAC for 4 RX channel operation. The AD9986 only has 2 physical ADCs. Internally, the physical ADCs are replicated such that 4 ADC channels' worth of data is sent over the JESD link to the FPGA.

The JESD204C modes define the parameters that are used to configure the JESD link. (see AD9986 Documentation for more details). The JESD link comprises 8 lanes operating at 16.22016Gbps each, in both the TX and RX direction. The JESD link conveys 4 DAC channels' information from the FPGA and conveys 4 ADC channels' information to the FPGA.

For this demonstration application, the FPGA receives all 4 ADC channels' information over the JESD. Logic inside the Hololink IP strips the ADC0 data from the incoming stream and sends only that data to the Host. In the TX direction, the Host sends one DAC channel's data to the FPGA. The FPGA replicates that one channel to all DAC channels before sending over the JESD link to the AD9986.

The clocking for this configuration is provided via the AD9986 EVAL board's HMC7044. The HMC7044 receives its input clock from an on-board oscillator. It then provides the necessary clocks to the AD9986 IC as well as the FPGA, via the FMC+ connector. The JESD operates in subclass 0 mode for this demonstration.

The current configuration of the AD9986 is set such that DAC channel 0 (DAC0) has a 0 MHz upconversion frequency. DAC channel 1 (DAC1) has an upconversion frequency of 1000 MHz. The ADC channel 0 (ADC0) has a downconversion frequency of 1000MHz.

1.2 System Requirements

HSB target host system – IGX Orin

Component	Minimum Requirement
OS	BaseOS v1.1.1
GPU	RTX 6000 Ada
Dependencies	Docker, CUDA Toolkit 12.X, NVIDIA Holoscan SDK 3.0, QEMU

1.3 Pre-Installation Checklist

☐ Host IGX Orin system installed with BaseOS v1.1 and configured with ConnectX, RDMA, and BAR1 set to 8GB

☐ Confirm GPU driver and CUDA are installed

☐ Docker installed and running

☐ Access to NVIDIA NGC registry (see end of host setup here)

☐ Access to HSB repository

☐ HSB 100G Board

☐ EVAL-AD9986 MxFE evaluation board from Analog Devices

☐ 2x 100 Gb/s Mellanox QSFP Cables, e.g. MCP1600-E001E30

☐ >=1ft SMA M-M RF coax cable, max frequency >= 8 GHz (For provided demonstration application)

☐ (Optional) GNU Radio installed for live signal monitoring (For provided demonstration application)

Chapter 2

Installation Steps

2.1 Hardware Setup

1. Follow **Figure 2-1** to connect the 2x QSFP cables to port J2 and J3 on the HSB board.

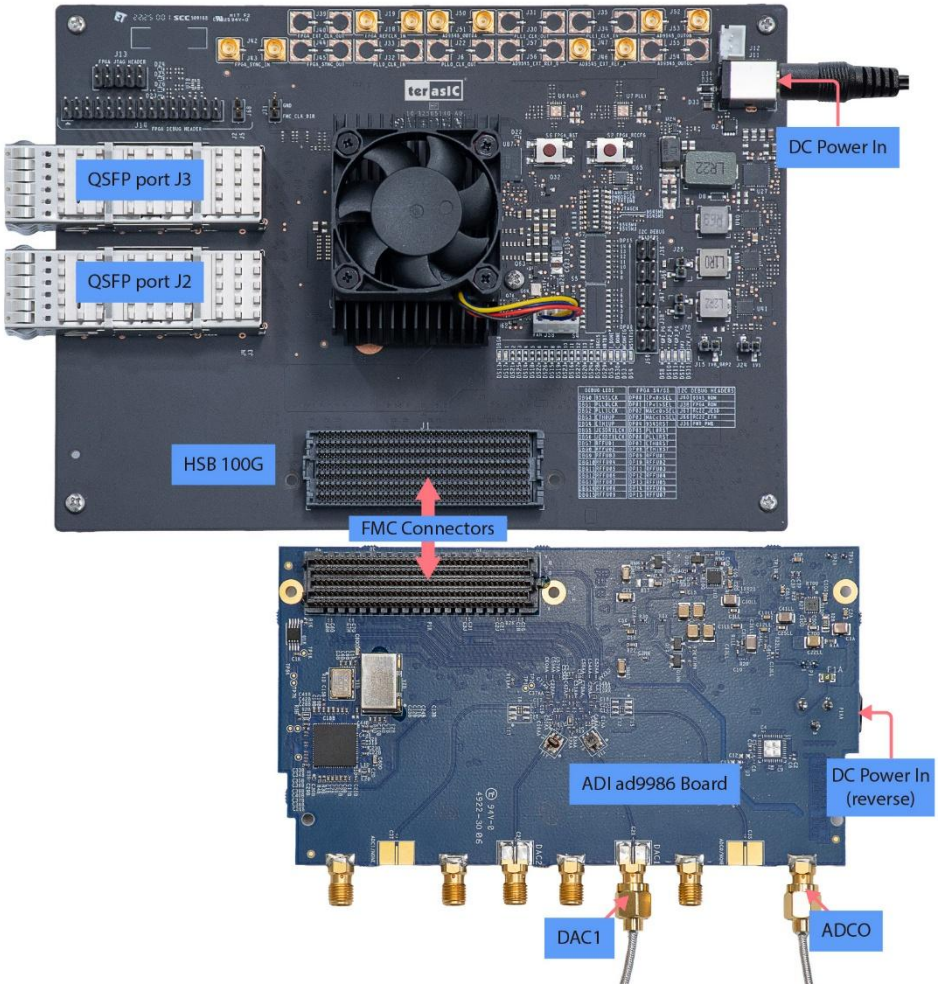


Figure 2-1 HSB 100G and AD9986 board with connection labels

2. For the signal generator demo application, connect the ADC0 and DAC1 on the ADI board using the RF coax cable.
3. Connect the QSFP cable from HSB port J3 to the port labelled 4 and HSB port J2 to the port labelled 3 on the back of the IGX as shown in **Figure 2-2**.

Figure 2-2 IGX Dev Kit backplane connections

4. Connect the FMC connectors on the HSB100G and the AD9986 board shown in **Figure 2-1**, mount the two together as shown in **Figure 2-3**, power on ADI board, and then HSB 100G.

Figure 2-3 HSB100G and AD9986 connected and powered

NOTE: When turning the HSB and AD9986 board ON or OFF, perform the operation on the AD9986 first, followed by the HSB.

2.2 Host Setup

Clone the NVIDIA holoscan sensor bridge repository from github:

$ git clone -b 2.2.0-EA https://github.com/nvidia-holoscan/holoscan-sensorbridge.git

Host setup should follow the published procedure of the 10G version of HSB once ConnectX, RDMA, and BAR1 have been set up per Pre-Installation Checklist.

Additionally, once the ConnectX ports are set up, ensure the MTU for each connection is at least 9000 to support RoCE packets.

Figure 2-4 ConnectX connection identities/settings on each QSFP port

2.3 Build the Virtual Machine

Since the signal generator example uses an Analog Devices MxFE, the virtual machine runs a Yocto image that is built using the Analog Devices Kernel along with extra Holoscan-provided drivers and device tree configuration that connects the MxFE (specifically, an AD9986 + HMC7044) to HSB. The meta-hsb directory defines a Yocto meta layer that provides all the code and recipes that are needed to build the HSB-enabled sidecar VM image using the following instructions.

1. Review the Yocto instructions to install required packages and set up the environment for buildi ng the Yocto image.
2. In the holoscan-sensor-bridge/configurator directory, clone the required Yocto layers:

   $ git clone -b scarthgap-5.0.9 git://git.yoctoproject.org/poky

   $ git clone -b scarthgap https://github.com/openembedded/metaopenembedded.git
3. Setup the build environment (this must be done in every shell that will be used to perform bitbake commands):

   $ source poky/oe-init-build-env

Note that when this is called, it will automatically create an empty project in the build directory and will change your path to that directory.

4. Add the absolute paths to the ***meta-openembedded/meta-oe*** and ***meta-hsb*** directories to the ***conf/bblayers.conf*** in the newly created project directory. The paths should be added to the end, after ***meta-yocto-bsp***, and need to be in that order (***meta-oe*** first followed by ***meta-hsb***). For example (change workspace with the path that contains your holoscan-sensor-bridge directory):

   BBLAYERS ?= " \

   /workspace/holoscan-sensor-bridge/configurator/poky/meta \

   /workspace/holoscan-sensor-bridge/configurator/poky/meta-poky \

   /workspace/holoscan-sensor-bridge/configurator/poky/meta-yocto-bsp \

   /workspace/holoscan-sensor-bridge/configurator/meta-openembedded/meta-oe \

   /workspace/holoscan-sensor-bridge/configurator/meta-hsb \

   "

5. Set the MACHINE variable in conf/local.conf to hsb-ad9986:

   $ echo 'MACHINE="hsb-ad9986"' >> conf/local.conf

6. Build the core-image-minimal image:

   $ bitbake core-image-minimal

Note: The default output directory for the image will be: tmp/deploy/images/hsb-ad9986.

Note: The build process is long and may take up to 1.5-2 hrs on an IGX.

2.4 Build and start HSB container

Use the build.sh script to automatically build the HSB demo with `--dgpu` for the host system's GPU configuration:

$ sh /path/to/holoscan-sensor-bridge/docker/build.sh --dgpu

To use the python interface, no further action is needed. To use the C++ applications or any related C++ components, build the signal_generator target in cmake which will compile all the dependencies and a binary in the build folder.

$ cd /path/to/holoscan-sensor-bridge

$ configurator/demo.sh /

$ mkdir build

$ cmake -S . -B build

$ cmake --build build -j --target signal_generator

2.5 Update the FPGA Firmware (as needed)

If newer firmware is available for the HSB 100G, perform the following steps.

Note: These steps are run inside the docker container (ie, the ***/path_to_holoscan-sensorbridge/configurator/demo.sh*** has been run.)

1. Retrieve .rpd file for the latest firmware and place it in **/path/to/holoscan-sensorbridge/scripts**.
2. Run the following sequence of commands assuming a firmware dated 06/09/2025:

   $ cd /path_to_holoscan-sensor-bridge/scripts

   $ python3 generate_manifest.py --version=250609 --manifest=manifest-hsb100g-250609.yaml --fpga-uuid=7a377bf7-76cb-4756-a4c5-7dddaed8354b –stratix-file=updated_firmware.rpd

   $ hololink program manifest-hsb100g-hsb100g-250609.yaml

The firmware update will take ~10 mins. Do not power off or disconnect any cables from the FPGA while reprogramming. After the programming completes, the new image will be configured automatically. Pressing the FPGA_RCFG button on the board or power cycling the board will also load the newlyprogrammed image.

2.6 Optional Components

Installation of GNURadio is provided in the **Chapter 7 Appendix**.

Chapter 3

Running the Demo

All steps below must be completed for each run of the application unless otherwise stated.

1. Ensure access control is enabled (Once after each system boot, before starting demo container):

   $ xhost +

2. Launch the container:

   $ sh /path/to/holoscan-sensor-bridge/configurator/demo.sh

   If the Yocto VM image (VM image directory) was copied or installed to a directory outside of `holoscan-sensor-bridge/configurator`, specify it explicitly with the following option appended to the command in step 2.

   --spi-vm=/path/to/target_directory

3. Confirm VM launch by checking the pid:

   $ cat /path/to/holoscan-sensor-bridge/spi-vm.pid

The live status of the VM can be monitored via the log file generated:

$ tail -f /path/to/holoscan-sensor-bridge/spi-vm.log

The VM is fully booted when the HSB-SPI Daemon connects to Hololink.

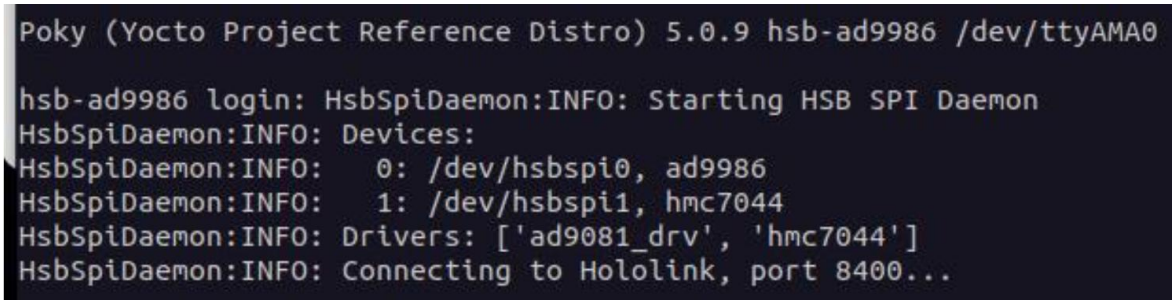


Figure 3-1 HSB-SPI Daemon fully started

It is recommended to wait for the VM to be fully booted before running applications using the ADI board otherwise errors might show up in launch of the SPI daemon.

4. Run the shell script launcher for signal_generator.py:

   $ examples/signal_generator.sh

5. The signal generator application opens a GUI that is used to control the signal generation and visualization for the demonstration. The application also configures the NVIDIA Holoscan sensor bridge and the AD9986 board for the demonstration. After a successful configuration, the GUI will look like the below:

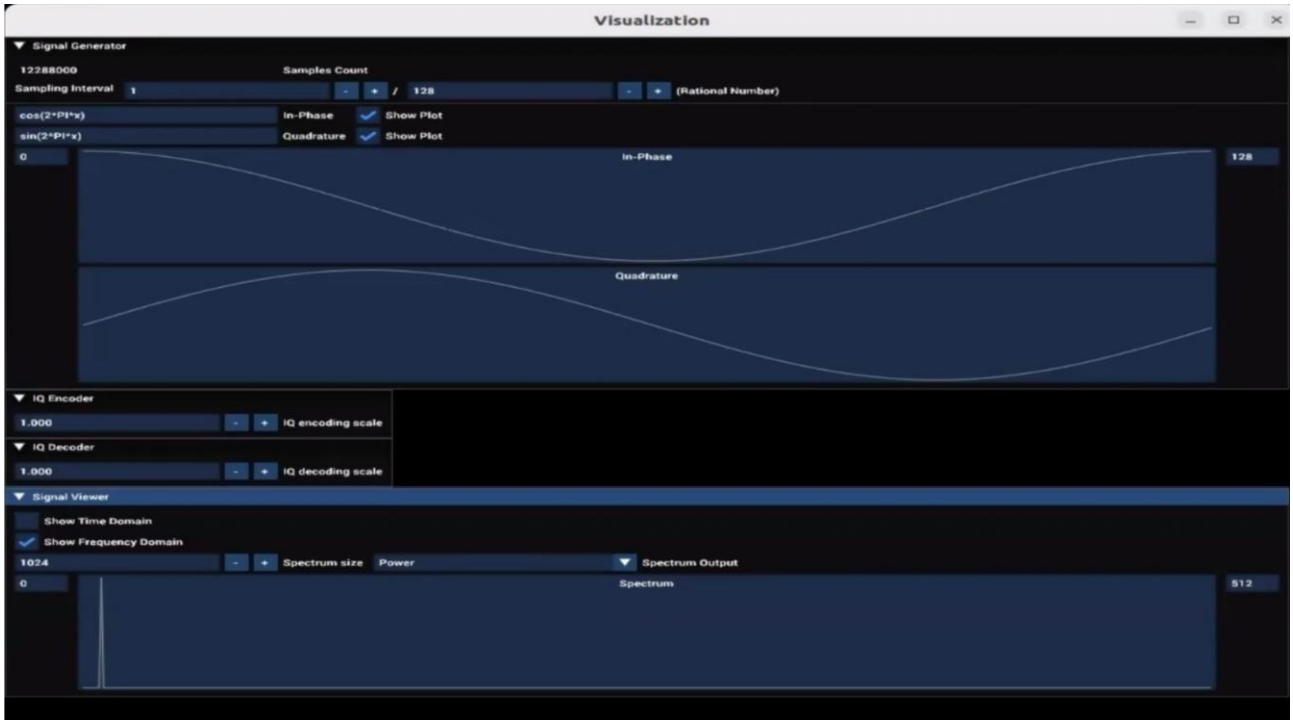


Figure 3-2 Signal Generator application interface

By default, the signal generator configures one complex sinusoid (one frequency tone) that is sent to the AD9986 via the HSB. The visualization is set up to show the power spectrum of the received data from the one ADC channel of the AD9986. The GUI functionality is explained in detail in Signal Generator/Viewer GUI.

3.1 GNU Radio (Optional)

To view live data from the receiver app outside of the User Interface, e.g. on a separate system, connect a separate process to the configured UDP IP/port while the demo application is running. For example, with GNU Radio:

1. Open a new terminal and run the following command in it:

   $ gnuradio-companion &

2. This will open a new, empty canvas.

   NOTE: there is a bug in GNU Radio so sometimes it opens and immediately close. If this happens, try a couple of times until the it no longer closes.

3. A sample gnuradio app is provided within the holoscan sensor bridge repository. Open /path/to/holoscan-sensor-bridge/examples/signal_generator_client.grc

   a. 983.04 Msps is the default configuration used in the demo application for the AD9986 board.

b. The default points sent per UDP datagram is 8192 points. This can be configured, but matching will lead to better app stability.

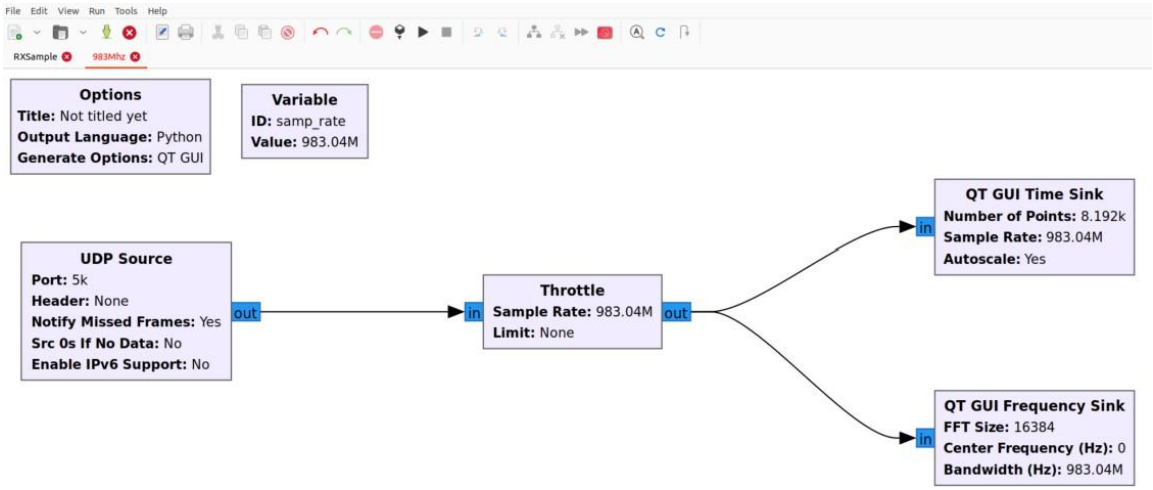


Figure 3-3 GNURadio sample application flow

4. Click the play button (Execute Flow Graph) to start receiving data while the Signal Generator app is running.

Chapter 4

Signal Generator / Viewer GUI

The signal generator and viewer GUI application is used to control and display the data sent to/received from the AD9986 via the HSB. The features of the application GUI will be explained below.

Figure 4-1 Signal Generator application running, sending multiple tones to HSB 100G + AD9986

The picture shows the GUI when configured to generate 3 complex sinusoidal tones with different frequencies and different amplitudes.

1. **Samples Count**: Number of samples generated by the signal generator per pipeline cycle. This number must be adequately sized (> 4M samples) to prevent underflow at the FPGA. This number cannot be changed in real time. The sample count must be divisible by 16. This is to ensure that Ethernet packets are sized to an integer number of 512-bits, which is the fundamental Ethernet data bus width in the FPGA.

2. **Sampling Interval**: Defines the values of the ***x*** variable when the signal expressions are evaluated. It is a rational number and both the numerator and the denominator can be set. This value determines the fundamental frequency of the generated signals. For example, the default setting

is 1/128. The sample rate of the DAC is 983.04MHz. The fundamental frequency is then 983.04e6 * (1/128) = 7.68MHz. It is recommended that the sampling interval is set for 1 / 2^n.

3. **Signal Equations**: Equations can be entered into the In-phase and Quadrature boxes to define the generated signal. The expression supports the 4 basic mathematical operations (+, -, *, /) including the plus and minus unary operations and parentheses (round brackets). One variable - ***x*** (lower case x) - is supported. The value it will be given during the evaluation processes is determined by the ***Sampling Interval*** field. The constant ***PI*** can be used as well as some common math functions like ***cos*** and ***sin***. It is recommended that the In-phase and Quadrature components are kept orthogonal to define complex sinusoids.
4. **Show Plot**: These checkboxes can be used to show/hide the time domain view of the generated signal components.
5. **Low Sample Range**: Used in conjunction with 6, defines the range of samples displayed in the signal windows.
6. **High Sample Range**: Used in conjunction with 5, defines the range of samples displayed in the signal windows.
7. **IQ Encoding Scale**: The encoding scale is used to adjust the input IQ samples, after generation, when mapped from floating point to integer. When the scale = 1, it is assumed that the signal values lie within the range of [-1, 1]. If the signal values lie outside this range, the scale needs to be increased (adds attenuation) so that the samples do not clip when mapped into the integer bit space. See Example Configuration for an example that manipulates this scaling value.
8. **IQ Decoding Scale**: This parameter adjusts the amplitude of the IQ samples received from JESD when mapping from integers to floating point. This value defaults to 1 so applies no scaling to the received data. Setting > 1 will increase the amplitude of the samples, and setting < 1 will decrease the amplitude of the samples.
9. **Show Time Domain/Show Frequency Domain**: These check boxes can be used to show/hide the received data signal time domain views and frequency domain view.
10. **Low Sample Range**: Used in conjunction with 11, defines the range of samples displayed in the time domain signal windows.
11. **High Sample Range**: Used in conjunction with 10, defines the range of samples displayed in the time domain signal windows.
12. **Spectrum Size**: Defines the number of points in the FFT that is used to display the received signal in the frequency domain. It is recommended that the spectrum size is a power of 2.
13. **Spectrum Output**: Select between power spectrum (complex I/Q amplitudes squared) and amplitude spectrum (complex I/Q amplitude).
14. **Low Sample Range**: Used in conjunction with 15, defines the range of FFT samples (bins) to display in the frequency domain window.
15. **High Sample Range**: Used in conjunction with 14, defines the range of FFT samples (bins) to display in the frequency domain window.

4.1 Example Configuration

The GUI picture above shows an example configuration setup to generate 3 complex sinusoids with different frequencies and amplitudes. Recall that the fundamental frequency is defined as 983.04e6 * sampling_interval = 983.04e6 * (1/128) = 7.68MHz.

The tones are defined using the In-Phase and Quadrature equations:

In-Phase: cos(2*PI*x)+0.5*sin(2*PI*4*x)-1.5*sin(2*PI*3*x)

Quadrature: sin(2*PI*x)-0.5*cos(2*PI*4*x)+1.5*cos(2*PI*3*x)

The first pair of In-Phase and Quadrature components defines a sinusoid with amplitude 1 at the fundamental frequency, 7.68MHz. The second pair defines a sinusoid with amplitude 0.5 at 4x the fundamental frequency, 30.72MHz. The third pair defines a sinusoid with amplitude 1.5 at 3x the fundamental frequency, 23.04MHz.

Recall that DAC1 is connected to ADC0 on the AD9986 via the SMA cable. The received data is displayed in the GUI. Hovering the mouse pointer over the tone peaks in the frequency domain display will show the FFT bin number. The FFT Spectrum size is set to 1024, so the bin resolution is 983.04e6 / 1024 = 960kHz. The first peak in the FFT display is ~bin 8; 8 * 960kHz = 7.68MHz. The second peak in the FFT display is ~bin 24, or 23.04Mhz. The third peak is ~bin 32, or 30.72MHz.

Note that the IQ encoding scale is set to 3.4. If set too low, the output samples are too large, clipping on the DAC output occurs, and data at the input to ADC0 is also clipped. The data displayed in the received time domain will also show the clipping. Increasing the encoder scaling will "attenuate" the transmitted signal such that no clipping occurs. The scaling should be adjusted based on the summation of the amplitudes of the generated signals. In the above example, that is: 1 + 0.5 + 1.5 = 3. Setting the scaling above at 3.4 reduces the overall amplitude further.

4.2 App Configuration

- App Users

Signal Generator User Interface

Command line arguments for configuring the user interface and application. See the ***/examples/signal_generator.sh*** for some usage of the below arguments.

Argument	Type	Example arg	Description
expression-i	String	"--expression- i=cos(2*PI*x)"	User-defined in-phase (i) and quadrature (q) signal components. Expressions support the followingoperators: +, -, *, / (floating division) One independent variable representing time: x Increment of variable x is determined by
expresion-q	String	"--expresion- q=sin(2*PI*x)"	

			`sampling-interval` argument Supported constants: `PI` = M_PI macro (*~3.14159265358979323846*)
no-gui	switch	--no-gui	Disable the graphic user interface (GUI). Default is GUI is enabled
real-time	switch	--real-time	Set process scheduler to real-time prioritization (POSIX). Default is not real-time
rx-hololink	String	--rx-hololink=192.168.0.2	IP address of HSB FPGA that the data is received from Default: "" (receiver is disabled in this process)
rx-ibv-name	String	--rx-ibv-name=mlx5_0	Local IB device name used for reception. Default is "mlx5_0"
rx-ibv-port	unsigned int	--rx-ibv-port-1	Port of the local IB device used for reception. Default is 1.
samples-count	unsigned int	--samples-count=12288000	The number of samples to generate per execution of signal generator operator. Note that this value has an impact on JESD channel overflow/underflow Default is 4096 * 3000 = 12,288,000 Requirements: must be multiple of 16 (to align to the fundamental 512-bit Ethernet data bus in the FPGA)
sampling-interval	Rational number as string	--sampling-interval=1/128	The time step to use for the independent variable in the `expression-i`/`expression-q` arguments. Default is 1/128, i.e. the difference between any two samples of `x` is 1/128~=0.0078125
tx-hololink	String	--tx-hololink=192.168.0.3	IP address of HSB FPGA that the data is received from Default: "" (transmitter is disabled in this process)
tx-ibv-name	String	--tx-ibv-name-mlx5_1	Local IB device name used for reception. Default is "mlx5_1"
tx-ibv-port	unsigned int	--tx-ibv-port-1	Port of the local IB device used for reception. Default is 1
tx-ibv-qp	unsigned	--tx-ibv-qp=2	QP number for the IBV stream that the

	int		data is transmitted to
tx-queue-size	uint64_t	--tx-queue-size=3	The number of buffers that can wait to be transmitted
udp-ip	String	--udp-ip=127.0.0.1	The IP address to transmit the received I/Q data to. Default is “”. Note: examples/signal_generator.sh will default to 127.0.0.1
udp-port	uint16_t	--udp-port=5000	The port to transmit received I/Q data to. Default is 5000

- App Developers

Operators

SignalGeneratorOp

Argument	Type	Example	Description
in_phase	String	Arg(“in_phase”, “cos(2*PI*x)”)	Expression to use to generate I/Q signal components. See `expression-i` and `expression-q` in Signal Generator User Interface for details
quadrature	String	Arg(“quadrature”, “sin(2*PI*x)”)	
renderer	ImGuiRenderer*	Arg(“renderer”, new hololink::ImGuiRenderer())	A pointer to an ImGuiRender. In the app defaults, this is just a `new ImGuiRenderer()` shared between all operators
samples_count	unsigned int	Arg(“samples_count”, 12288000)	The number of samples to generate per execution of signal generator operator. Note that this value has an impact on JESD channel overflow/underflow Default is 4096 * 3000 = 12,288,000 Requirements: must be multiple of 16 (to align to the fundamental Ethernet data bus in the FPGA)

sampling_interval	Rational (int, int)	Arg("sampling_interval", Rational(1, 128))	The time step to use for the independent variable in the `expression- i`/`expression- q` arguments. Default is 1/128, i.e. the difference between any two samples of `x` is 1/128~=0.0078125
cuda_toolkit_include_path	String	Arg("cuda_toolkit_include_path", "/usr/local/cuda-12")	The path of the CUDAToolKit installation. Default will use cmake to attempt to determine path at compile time

SignalViewerOp

Argument	Type	Example	Description
renderer	ImGuiRenderer*	Arg("rendered", new hololink::ImGuiRenderer())	A pointer to an ImGuiRender. In the app defaults, this is just a `new ImGuiRenderer()` shared between all operators

IQEncoderOp & IQDecoderOp

Argument	Type	Example	Description
renderer	ImGuiRenderer*	Arg("rendered", new hololink::ImGuiRenderer())	A pointer to an ImGuiRender. In the app defaults, this is just a `new ImGuiRenderer()` shared between all operators
Scale	float	Arg("scale", 1.0f)	

RoceTransmitterOp

Argument	Type	Example	Description
ibv_name	ImGuiRenderer*	Arg("ibv_name", "mlx5_0")	Local IB device name used for reception. Default is "roceP5p3s0f0"
ibv_port	uint32_t	Arg("ibv_port", 1u)	Port of the local IB device used for reception. Default is 1
hololink_ip	String	Arg("hololink_ip", "192.168.0.3")	HSB FPGA receiver address (host transmit target). Default is ""
ibv_qp	uint32_t	Arg("ibv_qp", 2u)	QP number for the IBV stream. Default is 2
buffer_size	uint64_t	Arg("buffer_size", 1024 * 1024)	The maximum buffer size in bytes for RoCE transmission. Default is 2^30
queue_size	uint64_t	Arg("queue_size", 1u)	The number of buffers that can wait to be transmitted. Default is 1
on_start*	void (const ConnectionInfo &)	Arg("on_start", std::function<void(const ConnectionInfo&)>())	Callback function to be called when the connection is created
on_stop*	void (const ConnectionInfo &)	Arg("on_stop", std::function<void(const ConnectionInfo&)>())	Callback function to be called when the connection is destroyed

*ConnectionInfo is a struct with at least 1 member of type "uint32_t" named "qp_num" corresponding to the IBV stream.

RoceReceiverOp

Argument	Type	Example	Description
frame_size	uint64_t	Arg("hololink_ip", 49152000)	The size of the buffer in bytes required to receive a message transmitted from RoceTransmitterOp.
hololink_ip	String	Arg("hololink_ip", "192.168.0.2")	HSB FPGA transmit address (host receiver target). Default is ""

host_pause_mapping	uint32_t	Arg("host_pause_mapping", 1)	Map from host interface name for IBV to FGPA transmit pause signal. Need to provide value consistent with `ibv_name`
ibv_name	ImGuiRenderer*	Arg("ibv_name", "mlx5_0")	Local IB device name used for reception. Default is "roceP5p3s0f0"
ibv_port	uint32_t	Arg("ibv_port", 1u)	Port of the local IB device used for reception. Default is 1

Chapter 5

Uninstallation

To remove all docker images for HSB.

$ docker rmi $(docker images -q hololink*)

To remove HSB, delete the docker image and then remove the local hololink git repository where it was cloned.

Chapter 6

Glossary

Term	Definition
RDMA	Remote Direct Memory Access – the method or ability of one device to gain access to the memory of another without copying via CPU instructions.
RoCE	RDMA over Converged Ethernet – protocol that allows RDMA to be performed over IP.

Chapter 7

Appendix

AD9986

EVAL-AD9986

GNU Radio

7.1 Installing GNU Radio on L4T Ubuntu

1. Add the gnuradio repo to **apt**:

   $ sudo add-apt-repository ppa:gnuradio/gnuradio-releases

2. Update **apt** and Install dependencies:

   $ sudo apt-get update

   $ sudo apt-get install xterm python3-packaging gnuradio

3. Edit the **grc.conf** file to add **xterm** as shown below on line 9:

   $ sudo gedit /etc/gnuradio/conf.d/grc.conf

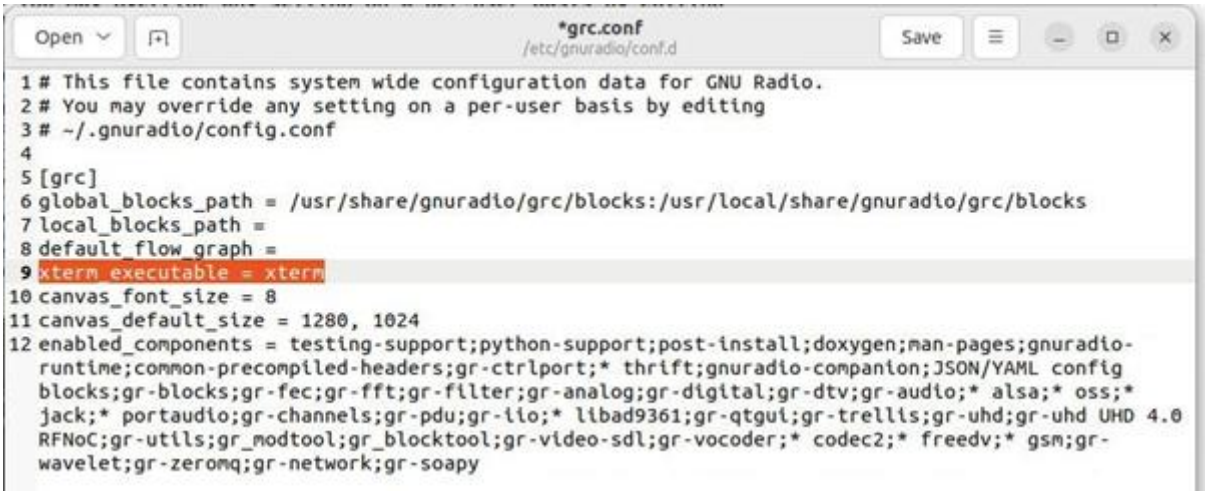


Figure 7-1 grc.conf modifications for GNURadio installation on L4T IGX BaseOS

4. Save grc.conf and close.

Chapter 8

Revision

8.1 Revision History

Date	*Version*	*Change Log*
2025.09	V1.0	Initial Draft
2025.09	V1.0.1	Review by N Team

8.2 Copyright Statement

Copyright © Terasic Inc. All Rights Reserved.