NI mmWave Instrument documentation

for Barkhausen Institut

written by NOFFZ Technologies, Vuk Obradovic, Peter Vago, 2020

1 Contents

Revisions

Date	Revision	Note	Author
14.5.2020.	1.0.0	First complete version	Vuk Obradović

Related documents

Document	Note
NI_mmWave_Getting_Started_Guide.pdf	Installation and connection guide
71_76_GHz_Millimeter_Wave_Transceiver_System.pdf	Datasheet
mmWave Transceiver System specifications.pdf	Detailed system specifications
NI-mmWave 18.1 Readme	NI mmWave driver readme

1 Introduction

mmWave instrument setup consists of two system used for research in field of mmWave RF communication and radar.

Systems are based on NI PXI mmWave modular hardware and LabVIEW software.

Purpose of setup is to allow arbitrary waveforms in mmWave range to be generated and measured through simple API.

Systems offer following main features:

- 3 modes of operation: mmWave RF, IF and baseband for both reception and transmission
- Arbitrary IQ waveform generation
- Configuration of RF parameters
- Configurable sampling frequency with independent settings for generation and acquisition
- Single burst and continuous generation modes
- Waveform playlist generation mode
- Arbitrary acquisition length
- Instrument control through local UI or remote Python API
- Independent and synchronized modes of operation
- Phase/Sync synchronization between systems
- Simulation mode for testing and experimenting with API without hardware

1.1 Installation

1.1.1 Hardware

System A should be assembled as described in following sections of "NI_mmWave_Getting_Started_Guide.pdf":

- **Baseband and IF:** System Setup -> Bidirectional system setup -> Interconnecting the Bidirectional SISO (Baseband and IF) Modules (page 29)
- **Radio heads:** System Setup -> Connecting mmWave Radio Heads to the System -> Connecting mmWave Radio Heads to a Bidirectional System -> Connecting mmRH-3647/3657 Radio Heads to a Bidirectional System (page 49)

System B should be assembled as described in following sections of "NI_mmWave_Getting_Started_Guide.pdf":

- **Baseband and IF:** System Setup -> Unidirectional system setup -> Interconnecting the Unidirectional SISO (Baseband and IF) Modules (page 15)
- **Radio heads:** System Setup -> Connecting mmWave Radio Heads to the System -> Connecting mmWave Radio Heads to a Unidirectional System -> Connecting mmRH-3647/3657 Radio Heads to a Unidirectional System (page 42)

System A and system B can be interconnected to allow trigger and LO sharing. More details are provided in chapter "2.4 System interconnection"

1.1.2 Software

Both systems come with preinstalled applications for operations. Application software automatically starts on system power up.

System requirements:

- RAM—8 GB RAM
- A screen resolution of 1,024 x 768
- Windows 10/8.1/7 64-bit with all available critical updates and service packs

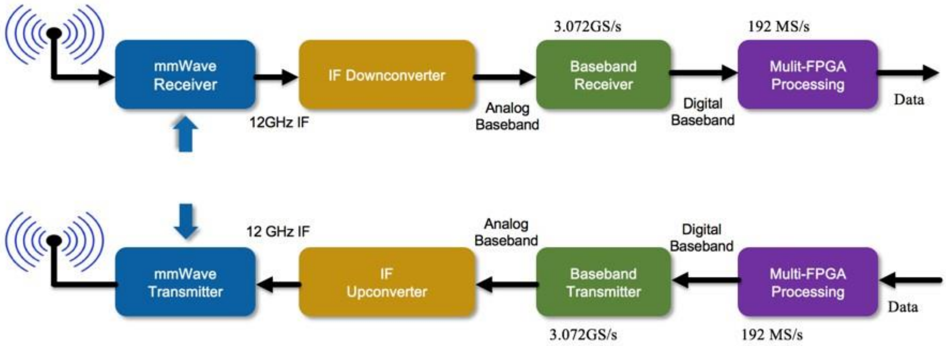
For installing software on other PC to use in simulated mode complete following steps:

- Download latest **NI Package manager** (https://www.ni.com/en-us/support/downloads/software-products/download.package-manager.html)
- Install package: **barkhausen-mmwave-radar_version_windows_x64.nipkg**

Detailed information regarding software can be found in chapter “4 Software”.
Development software list is given in chapter APENDIX 1

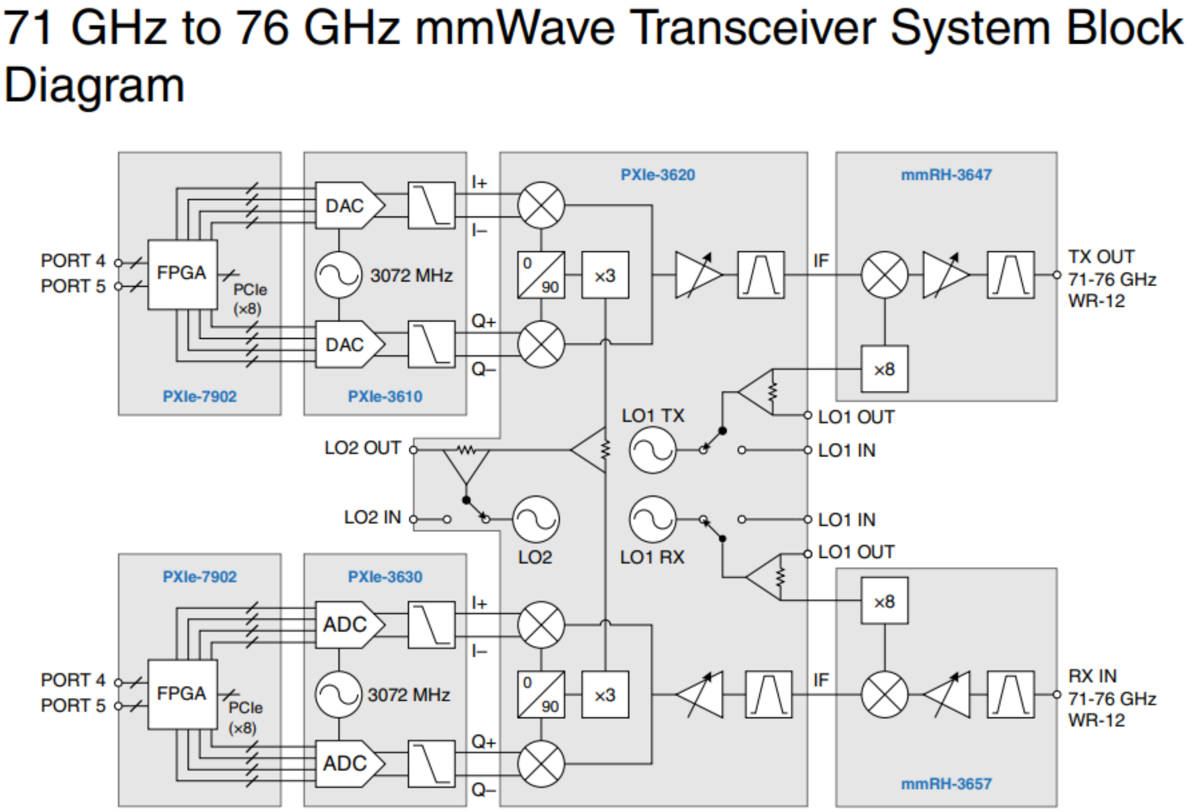
2 Hardware

System is based on 2GHz bandwidth NI PXI mmWave setup.



Picture 2-1 NI mmWave system

System is made of several blocks which allows flexibility in using different RF front ends or using IF only or baseband signals.

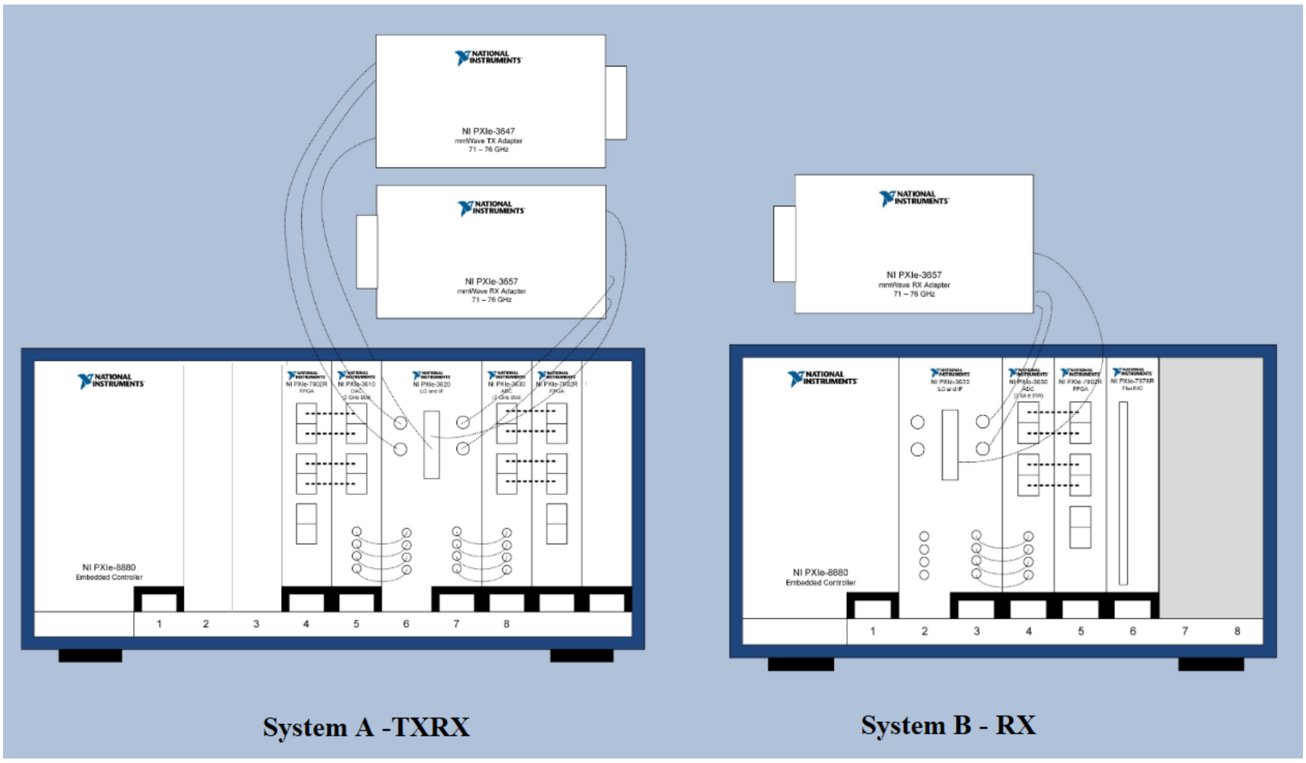


Picture 2-2 NI mmWave block diagram

System A is transceiver which supports coherent transmission and reception of RF signal with custom IQ samples.
System B is receiver which supports reception of RF signals.
Both systems have identical receiver hardware.

System can work on their own or they can apply following synchronization options::

- **Synchronized timing**: Systems have shared start trigger for transmission and both recievers.
- **Synchronized LO (coherent operation):** Local oscillator for receiver chain are shared between systems.



Picture 2-3 mmWave setup System setup

2.1 Characteristics

System characteristics:

Parameter	Value
Operational frequency	71-76GHz
Real-time bandwidth	2GHz
Interface	WR-12
Analog gain range	55dB
1dB Gain Compression	-12dBm
Noise Figure	6dB
IQ sampling rate	100K-3.072GS/s
IQ resolution	12 bits
IQ burst length	208n-100ms

Table 2-1 Receiver characteristics

Parameter	Value
Operational frequency	71-76GHz
Saturated power	24dBm
Interface	WR-12
Analog gain range	55dB
Output IP3	+30dbm
Real-time bandwidth	2GHz
IQ sampling rate	100K-3.072GS/s

IQ resolution	12 bits
IQ burst length	208n-100ms
Supported generation modes	Burst Continuous

Table 2-2 Transmitter characteristics

Parameter	Value
Trigger standard deviation (once system is initialized)	<1ns
Trigger standard deviation (between system resets)	<10ns
LO1 sharing max distance	
LO2 sharing max distance	

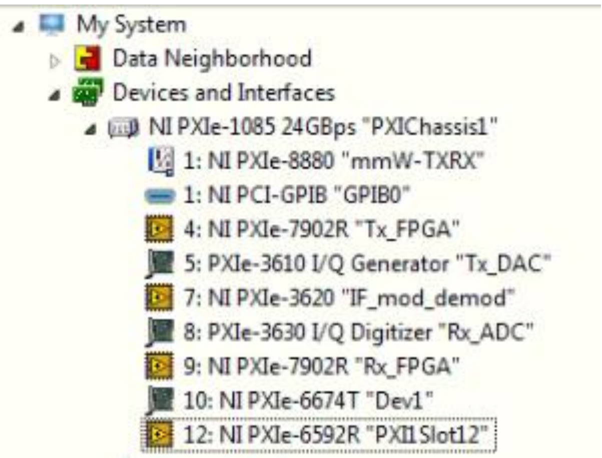
Table 2-3 Synchronization characteristics

2.2 System A

System A is Transceiver which consists of:

- PXI chassis with its modules
- TX mmWave RF front end
- RX mmWave RF front end

2.2.1 PXI system



```
My System
  Data Neighborhood
  Devices and Interfaces
    NI PXIe-1085 24GBps "PXIChassis1"
      1: NI PXIe-8880 "mmW-TXRX"
      1: NI PCI-GPIB "GPIB0"
      4: NI PXIe-7902R "Tx_FPGA"
      5: PXIe-3610 I/Q Generator "Tx_DAC"
      7: NI PXIe-3620 "IF_mod_demod"
      8: PXIe-3630 I/Q Digitizer "Rx_ADC"
      9: NI PXIe-7902R "Rx_FPGA"
      10: NI PXIe-6674T "Dev1"
      12: NI PXIe-6592R "PXI1Slot12"
```

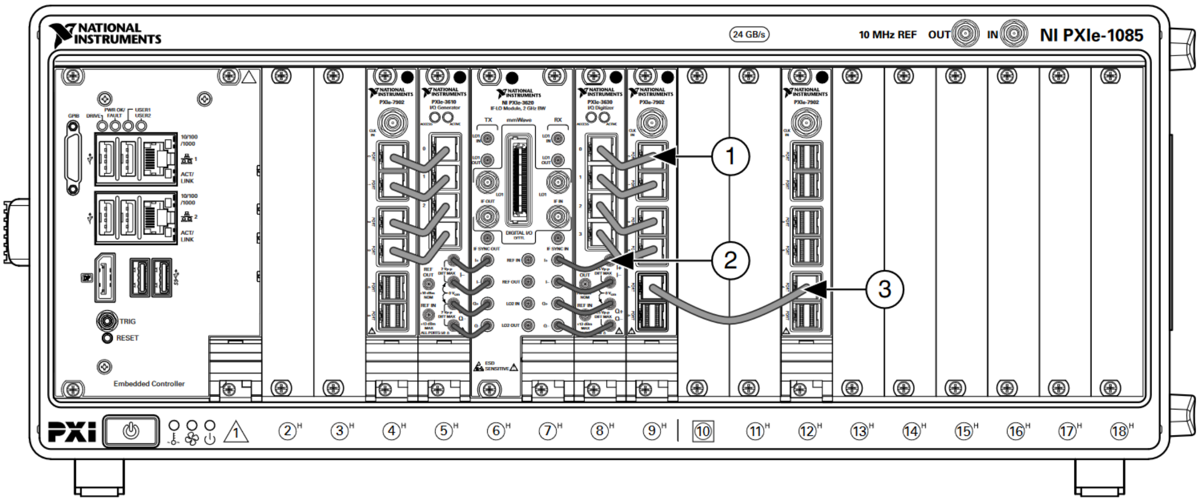
Picture 2-4 System A configuration

System modules:

Module	Slot	Name	Description
NI PXIe-1085 24GBps		PXIChassis1	PXI chassis which holds modules and provides trigger and clock synchronization between them
NI PXIe-8880	1	mmW-TXRX	PXI controller with Windows OS.
NI PXIe-7902R	4	Tx_FPGA	FPGA module with High speed serial interface. Transmitter digital baseband module which generates digital high speed IQ signals.
NI PXIe-3610 I/Q Generator	5	Tx_DAC	DAC converter for Tx baseband signal. Converts high speed serial digital IQ samples

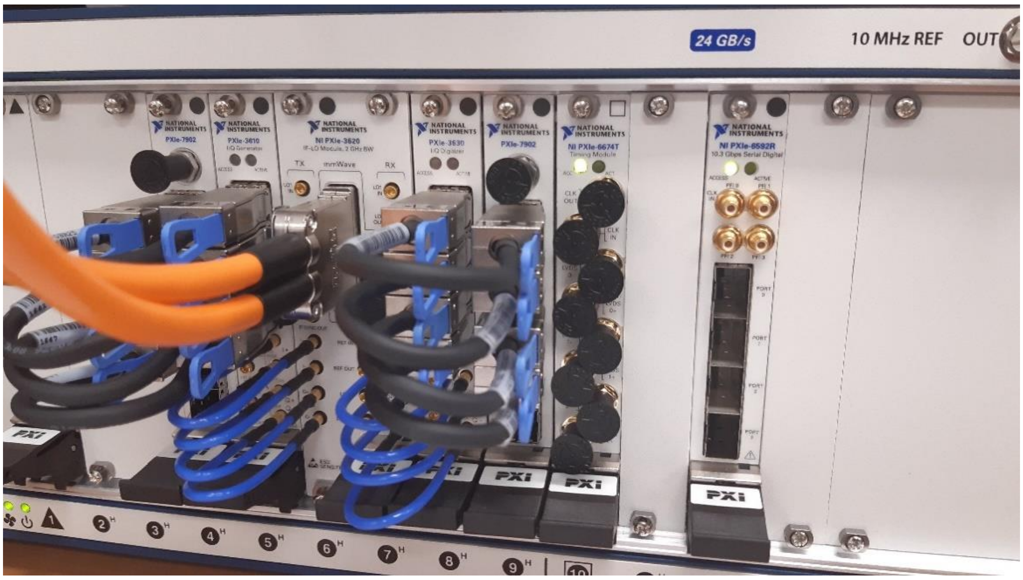
			into differential baseband I/Q samples with 3.072GS/s
NI PXIe-3620	7	IF_mod_demod	IF upconverter and downconverter. Modulates TX baseband signal to IF and demodulates RX IF signal to baseband. Provides LO for TX and RX RF radio heads.
NI PXIe-3630 I/Q Digitizer	8	Rx_ADC	ADC converter for baseband RX signal. Converts differential I/Q samples into high speed serial digital IQ data with 3.072GS/s
NI PXIe-7902R	9	Rx_FPGA	FPGA module with High speed serial interface. Reciever digital baseband module which receives digital IQ samples.
NI PXIe-6674T	10	TM	Timing module. Provides more precise 100MHz reference clocks to PXI chassis. Allows exporting and importing of trigger lines from PXI backplane.
NI PXIe-6592R	12	PXI1Slot12	*Unused* Additional FPGA module with serial interface for data processing

Figure 23. Bidirectional SISO (Baseband and IF) Chassis



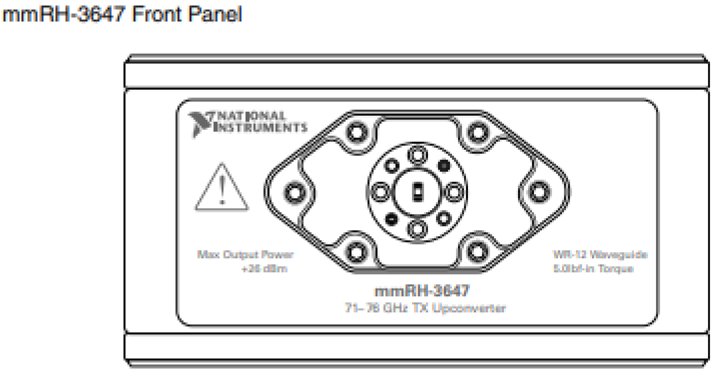
Picture 2-5 System A PXI (1)

]



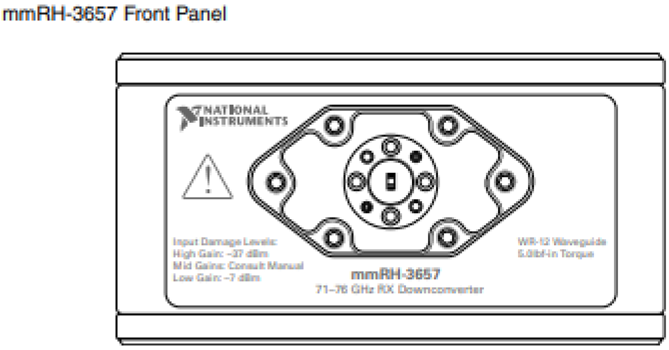
Picture 2-6 System A PXI (2)

2.2.2 TX mmWave RF front end



Picture 2-7 mmWave Radio head mmRH 3647 TX upconverter

2.2.3 RX mmWave RF front end



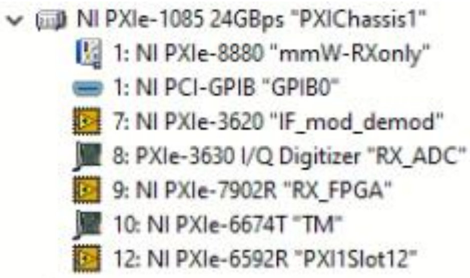
Picture 2-8 mmWave Radio head mmRH 3657 RX downconverter

2.3 System B

System B is Receiver which consists of

- PXI chassis with its modules
- RX mmWave RF front end

2.3.1 PXI system



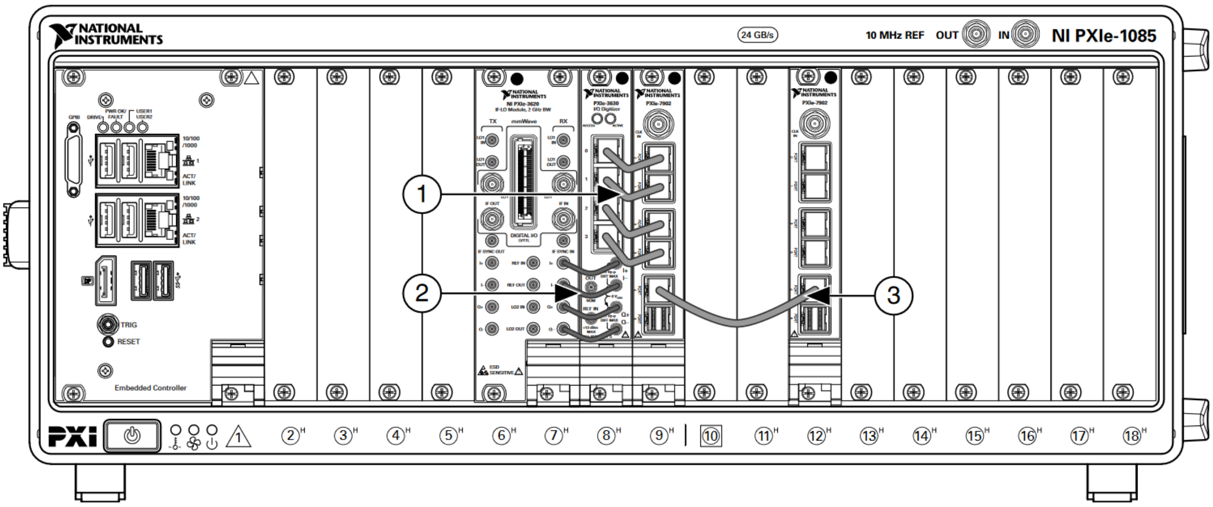
NI PXIe-1085 24GBps "PXIChassis1"
1: NI PXIe-8880 "mmW-RXonly"
1: NI PCI-GPIB "GPIB0"
7: NI PXIe-3620 "IF_mod_demod"
8: PXIe-3630 I/Q Digitizer "RX_ADC"
9: NI PXIe-7902R "RX_FPGA"
10: NI PXIe-6674T "TM"
12: NI PXIe-6592R "PXI1Slot12"

Picture 2-9 System B configuration

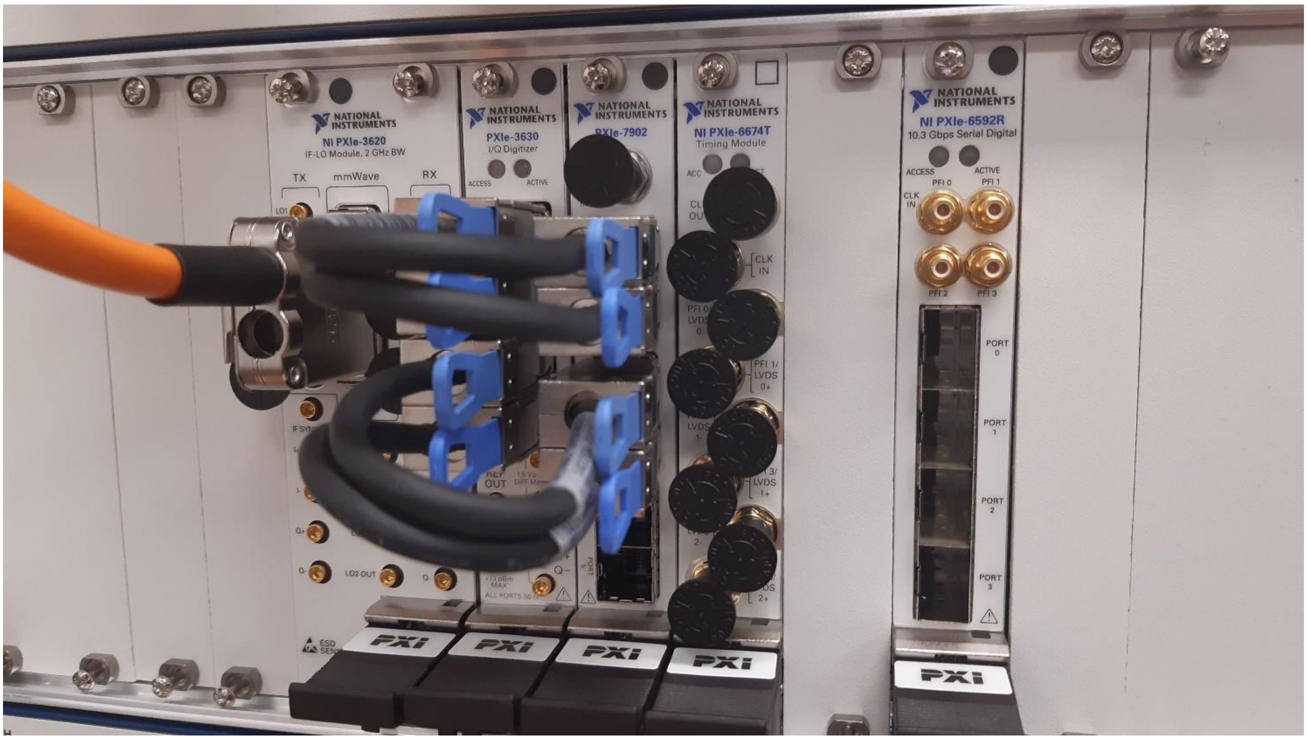
System modules:

Module	Slot	Name	Description
NI PXIe-1085 24GBps		PXIChassis1	PXI chassis which holds modules and provides trigger and clock synchronization between them
NI PXIe-8880	1	mmW-TXRX	PXI controller with Windows OS.
NI PXIe-3620	7	IF_mod_demod	IF upconverter and downconverter. Demodulates RX IF signal to baseband. Provides LO for TX and RX RF radio heads.
NI PXIe-3630 I/Q Digitizer	8	Rx_ADC	ADC converter for baseband RX signal. Converts differential I/Q samples into high speed serial digital IQ data with 3.072GS/s
NI PXIe-7902R	9	Rx_FPGA	FPGA module with High speed serial interface. Reciever digital baseband module which receives digital IQ samples.
NI PXIe-6674T	10	TM	Timing module. Provides more precise 100MHz reference clocks to PXI chassis. Allows exporting and importing of trigger lines from PXI backplane.
NI PXIe-6592R	12	PXI1Slot12	*Unused* Additional FPGA module with serial interface for data processing

Figure 9. Unidirectional SISO (Baseband and IF) RX Chassis



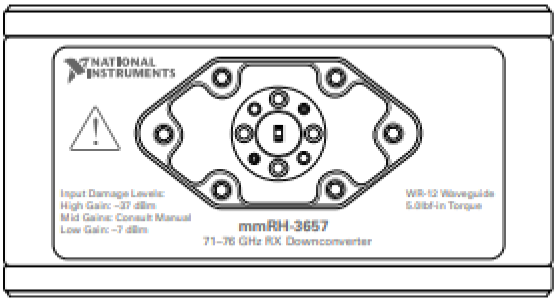
Picture 2-10 System B PXI (1)



Picture 2-11 System B PXI (2)

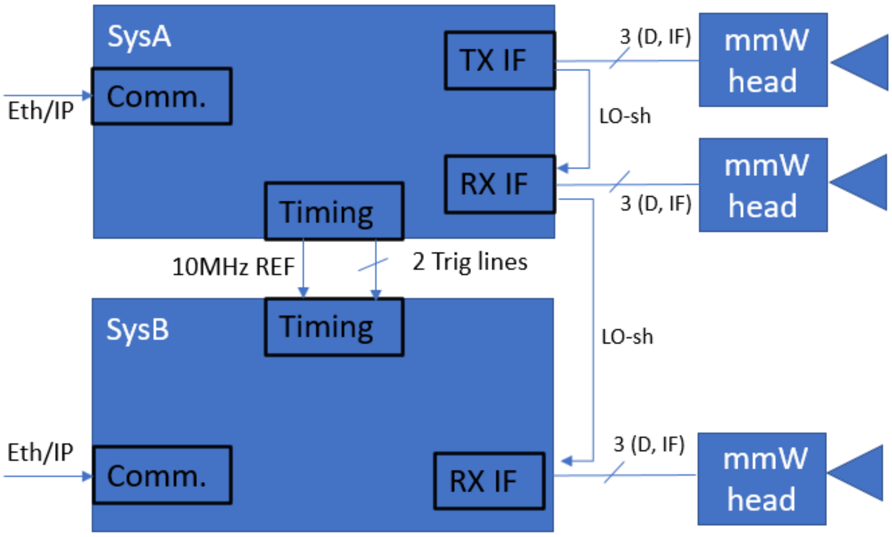
2.3.2 RX mmWave RF front end

mmRH-3657 Front Panel



Picture 2-12 mmWave Radio head mmRH 3657 RX downconverter

2.4 System interconnection



Picture 2-13 System interconnection diagram

Systems have following signals interconnected:

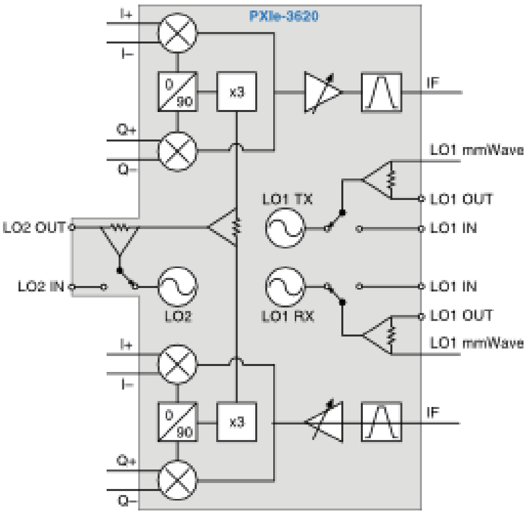
- **10MHz REF:** PXI Chassis reference clock, sharing of this clock provides in phase operation of all digital subsystems
- **Trigger lines:** PXI Trig line. Shared through timing module. Triger line sharing enables synchronous start of operation when generating and acquiring bursts.
- **Local oscillators:** IF module LOs. LO sharing enables coherent acquisition (with constant phase shift) on both systems

2.4.1 Trigger and Clock sharing

To use trigger and clock sharing make following connections:

- Connect **10MHz REF OUT** of System A chassis to **10MHZ REF IN** of System B
- Connect **PFI2** port of Timing module on System A to **PFI2** port of Timing module on System B

2.4.2 IF LO sharing



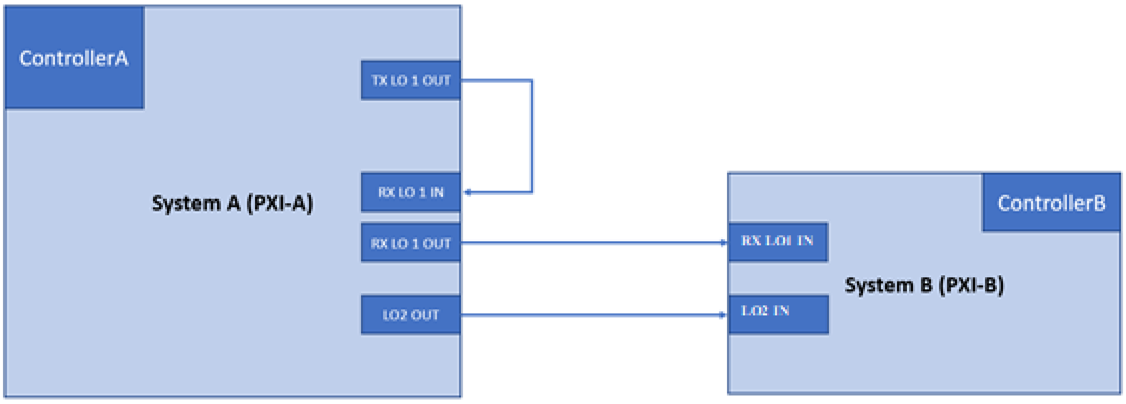
Picture 2-14 PXIe-3620 IF module

IF module has 3 Locals oscillators:

- **LO1 TX** – oscillator used by TX radio head. From this LO radio head creates carrier for RF upconversion.
- **LO1 RX** - oscillator used by RX radio head. From this LO radio head creates carrier for RF downconversion.
- **LO2** – oscillator used for IF up and downconversion

On System A LO1 TX is connected to LO1 RX to ensure coherent transmission and reception.

System A **LO1 RX** and **LO2** are connected to System B **LO1 RX** and **LO 2** to allow coherent operation of System A and System B receivers.



Picture 2-15 LO sharing details

LO1 and LO2 values are calculated based on desired IF/RF frequency.

If System is working as mmWave RF transceiver LO1 and LO2 frequencies depend on which RF heads are used:

Devices	LO Injection	RF Frequency	Formulas
mmRH-3602/3642/3652 mmWave Radio Heads	High-side	*RF Frequency = LO Frequency - IF Frequency*	*RH LO Frequency = LO1 Frequency* × 4 *IF Frequency = LO2 Frequency* × 3
mmRH-3603/3643/3653 mmWave Radio Heads	Low-side	*RF Frequency = LO Frequency + IF Frequency*	*RH LO Frequency = LO1 Frequency* × 3 *IF Frequency = LO2 Frequency* × 3
mmRH-3647/3657 mmWave Radio Heads	Low-side	*RF Frequency = LO Frequency + IF Frequency*	*RH LO Frequency = LO1 Frequency* × 8 *IF Frequency = LO2 Frequency* × 3

Table 2-4 Radio head LO and IF frequency calculation

In following table several examples for different RH are given:

mmWave Radio Heads	Mode	RF Frequency	IF Frequency	LO1 Frequency	LO2 Frequency
Any/none	IF only	—	11 GHz	—	3.667 GHz
mmRH-3602/3642/3652	mmWave	28.5 GHz	10.56 GHz	9.765 GHz	3.52 GHz (default)
mmRH-3603/3643/3653	mmWave	40 GHz	10.56 GHz	9.813 GHz	3.52 GHz (default)
mmRH-3647/3657	mmWave	73 GHz	12 GHz	7.625 GHz	4 GHz (default)

Table 2-5 LO1 and LO2 frequency examples

3 Instrument Operation

Instrument is controlled via “instrument app” on its Windows controller.

Instrument can have several different modes of operation and during operation it can go through several states.

3.1 Modes of operation

Software for mmW instrument was created to support following modes:

- **RF (default mode):** Complete chain is used.
    - TX: baseband signal is generated with DAC module, modulated to IF using IF module and further modulated to RF mmW band using RF head. TX signal is generated at RF head antenna.
    - RX: incoming RF mmW band signal is downconverted to IF using RF head, demodulated from IF to baseband using IF module and digitalized with ADC module. RX signal is measured at RF head antenna
    - User can set RF frequency and gains for TX and RX path
    - User can provide custom IQ signal for generation and setup custom acquisition parameters
- **IF:** NI mmW RF heads not used.
    - TX: baseband signal is generated with DAC module, modulated to IF using IF module. TX signal is generated at IF out port of “IF_mod_demod” module.
    - RX: incoming IF signal is demodulated from IF to baseband using IF module and digitalized with ADC module. RX signal is measured at IF IN port of “IF_mod_demod” module.
    - User can set IF frequency and gains for TX and RX path
    - User can provide custom IQ signal for generation and setup custom acquisition parameters
- **BB (Baseband):** Only baseband modules are used.
    - TX: baseband signal is generated with DAC module. TX signal is generated as two differential signals (I and Q) at ports I+/I-/Q+/Q- of “Tx_DAC” module.
    - RX: incoming baseband signal is digitalized with ADC module. RX signal is measured as two differential signals at ports I+/I-/Q+/Q- of “Rx_ADC” module.
    - RF configurations are ignored.
    - User can provide custom IQ signal for generation and setup custom acquisition parameters.

3.2 Instrument states

mmW instrument operates through several states. State transitions can be automatic or forced via API commands.

Instrument flows through following states:
- Off
- Initialization
- Configuration
- Calibration
- Running

Off

Default state at instrument app start. No instrument sessions are running, no FPGA bitfile is deployed.

Initialization

During this state instrument sessions are started and FPGA bitfiles are being deployed. At end of this state internal timing calibrations are executed (NI T-clock) to align timing on all modules on same chassis. Which instrument sessions are initialized depends on instrument setup (System A or System B) and operation mode (RF/IF/BB).

Following sessions are initialized:
- FPGA session for “Tx_FPGA”. Bitfile deployed: “TX_FPGA_driver_DRAM_v_1_2_1.lvbitx” (System A only)
- Session for controlling “Tx_DAC” module (System A only)
- Session for controlling “IF_mod_demod” module (IF and RF modes)

- FPGA session for “Rx_FPGA”. Bitfile deployed: “RX_FPGA_driver_DRAM_v_1_2.lvbitx”
- Session for controlling “Rx_ADC” module
- Session for controlling “TM” – timing module

Configuration

In this state instrument accepts command for changing various parameters and applies them. Parameters can be changed through remote control API or through instrument app local GUI.

Following configuration commands are available:

- **Configure RF:** Configures RF/IF frequency and gain for specified port (“tx” or “rx”)
- **Configure Fs:** Configures baseband sampling frequency for IQ samples for specified port (“tx” or “rx”). TX Fs specifies sampling frequency for provided TX burst samples. RX Fs specifies with which sampling frequency will be RX IQ samples be measured.
- **Equalization enable:** Enables or disabled equalization FIR filter for RX signal
- **Trigger sharing enable:** Enables or disabled trigger sharing between two Systems – timing synchronization.
- **LO sharing enable:** Enables or disables LO1 and LO2 sharing between two systems - phase/sync synchronization.

Calibration

Calibration state is used to obtain automatic equalization coefficients. In this state calibration signal is generated at TX port and acquired on RX port(s). Details regarding equalization will be provided in chapter “3.4 Calibration”

Running

In running state system is prepared to accept start triggers. On start trigger system starts generating preloaded TX IQ samples and starts acquisition of RX IQ samples.

When systems are running independently (trigger sharing is disabled) each system accepts its own software trigger (command given through API or UI).

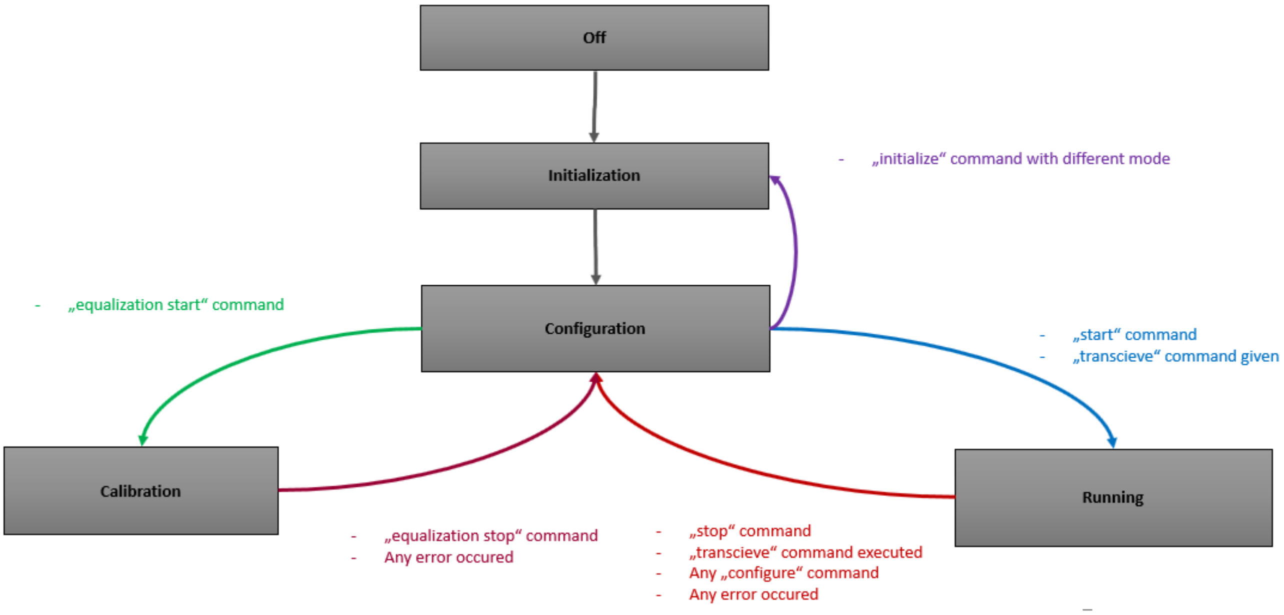
When systems are running with timing synchronization (trigger sharing is enabled) One system (System A) accepts its own software trigger (via command) and then distributes start trigger to other system (System B).

When system is in **Running** state several triggers can be sent without reloading TX IQ samples. Each time same IQ samples will be generated.

Details regarding TX samples generation and RX samples acquisition will be provided in chapters “3.5TX signal generation” and “3.6 RX signal acquisition”

3.2.1 State transitions

Following chart illustrates state transitions


Off
Initialization
Configuration
Calibration
Running
- „initialize“ command with different mode
- „equalization start“ command
- „start“ command
- „transcieve“ command given
- „equalization stop“ command
- Any error occured
- „stop“ command
- „transcieve“ command executed
- Any „configure“ command
- Any error occured

Picture 3-1 Instrument state transitions

- Instrument goes from **Off** to **Initialization** state automatically when application is started
- Every time when **Initialization** is complete system goes into **Configuration** state
- Issuing command which changes operation mode (example: system is in RF mode and command was given to change to IF) will trigger **Initialization** state again to change mode
- **Calibration** is entered with command to start equalization and exited with command to stop equalization
- **Start** command will put system in **Running** state
- Any configuration command (RF, Fs) will put system back in **Configuration** state
- Sending new start command with different arguments will apply those arguments and return system in Running state
- Whenever any error occurs in **Running** or **Calibration** state system returns to **Configuration** state.
- **Transcieve** command puts system in **Running** state and once completed returns system in **Configuration** state.

3.3 Commands

Instrument accepts following commands. Commands can be given through API or local GUI

Command	Input	Output	Valid state	Description
Initialize	• mode: RF/IF/BB		Configuration Off	Initializes system in given operation mode
Configure RF	• port: “tx”/” rx” • gain • frequency		Configuration Running	Configuration command. Configures RF port properties: gain and frequency. Depending on mode either IF or RF frequency is configured.
Configure Fs	• port: “tx”/” rx” • fs		Configuration Running	Configuration command. Configure port sampling frequency. Tx and rx sampling frequencies can be independently configured.
Trigger synchronization enable	• Enable: True/False		Configuration Running	Configuration command. Enables or disables start trigger sharing between systems.
LO synchronization enable	• Enable: True/False		Configuration Running	Configuration command. Enables or disables local oscillators (LO1 and LO2) sharing between systems.
Equalization enable	• Enable: True/False		Configuration Running	Configuration command. Enables or disables rx signal equalization filter.
Write TX	• Burst name • TX samples		Any	Low-level command Send TX IQ samples to instrument. Bursts are held in memory under given name until start command is received.
Start	• Burst playlist • Acquisition duration		Configuration Running	Low-level command Puts system in Running state. Burst playlist is list of bursts to be generated one after another. Bursts must be previously loaded using **Write TX** command. **Acquisition duration** specifies duration of RX IQ signal acquisition. Once system is in Running state it is ready to accept trigger. If start command is send while system was in Running state it will reload new playlist and reset Running state.
Send trigger	• Burst mode: burst/continuous		Running	Low-level command Send software trigger to instrument. This trigger will initiate tx signal generation and rx signal acquisition. Generation can be in burst or continuous mode
Fetch		• RX samples	Running	Low-level command Retrieves RX IQ samples from previously triggered acquisition.
Stop			Running	Low-level command Switches instrument from Running to Configuration state
Transceive	• Burst mode • Acquisition duration • TX samples	• RX samples	Configuration Running	High-level command. Integrates write tx, start, send trigger. Fetch and stop commands into single

				command. It supports only single burst generation in given mode.
Equalization start			Configuration	Calibration command. Starts Calibration state. Starts generation of calibration signal, calculates channel response and equalization filer coefficients. System will keep generating calibration signal while in Calibration state.
Equalization stop			Calibration	Calibration command. Stops calibration state. Stops calibration signal generation
Shutdown				Stop instrument application and switches system in Off state

3.4 Calibration

Calibration process is used to estimate channel response as freuquency response between generated IQ signal at DAC and measured IQ signal at ADC. It estimates therefore complete path (DAC + TX IF + TX RF + signal propagation through waveguide/antenna/air + RX RF + RX IF + ADC)

Estimated channel is therefore dependent on currently used RF settings (Frequency, tx and rx gains).

When Transceiver (System A) start equalization, it starts generating calibration waveform $Cal_{TX}(t)$ on its DAC.

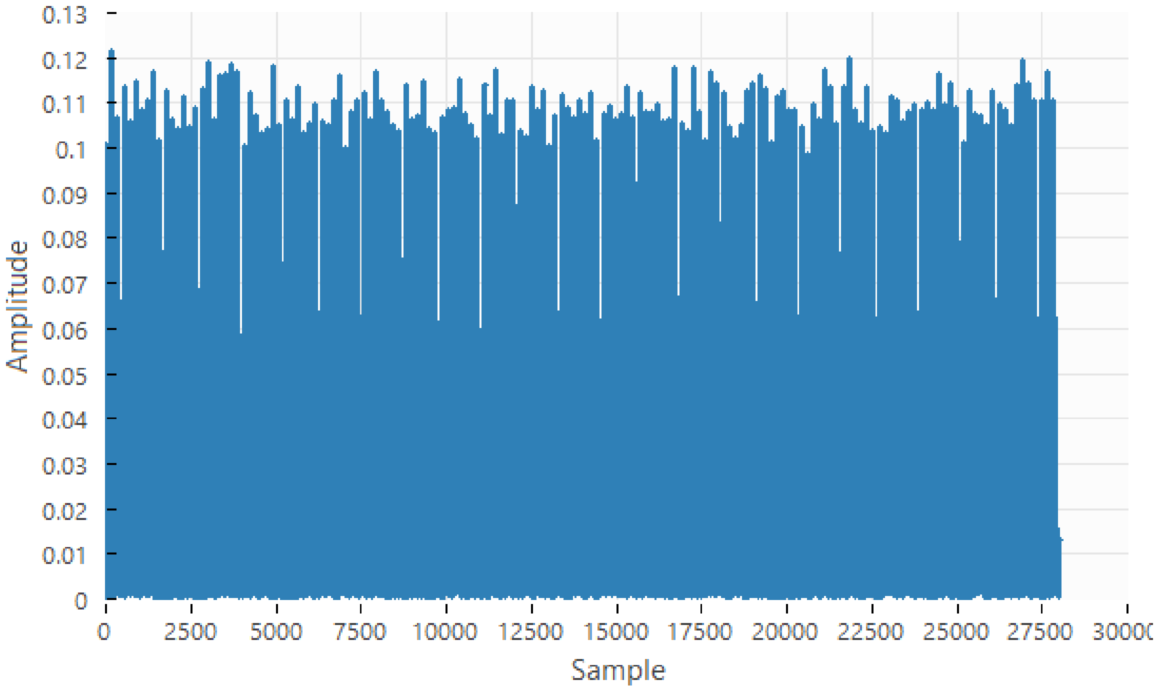
Calibration signal is train of high frequency pulses with following properties:

- Sampling frequency: max (3.072GS/s)
- Pulse duration: 1 sample (325.5ps)
- Pulse repetition frequency: 160 samples (52.083ns)
- Pulse amplitude: 0.5 (half of maximum DAC range)

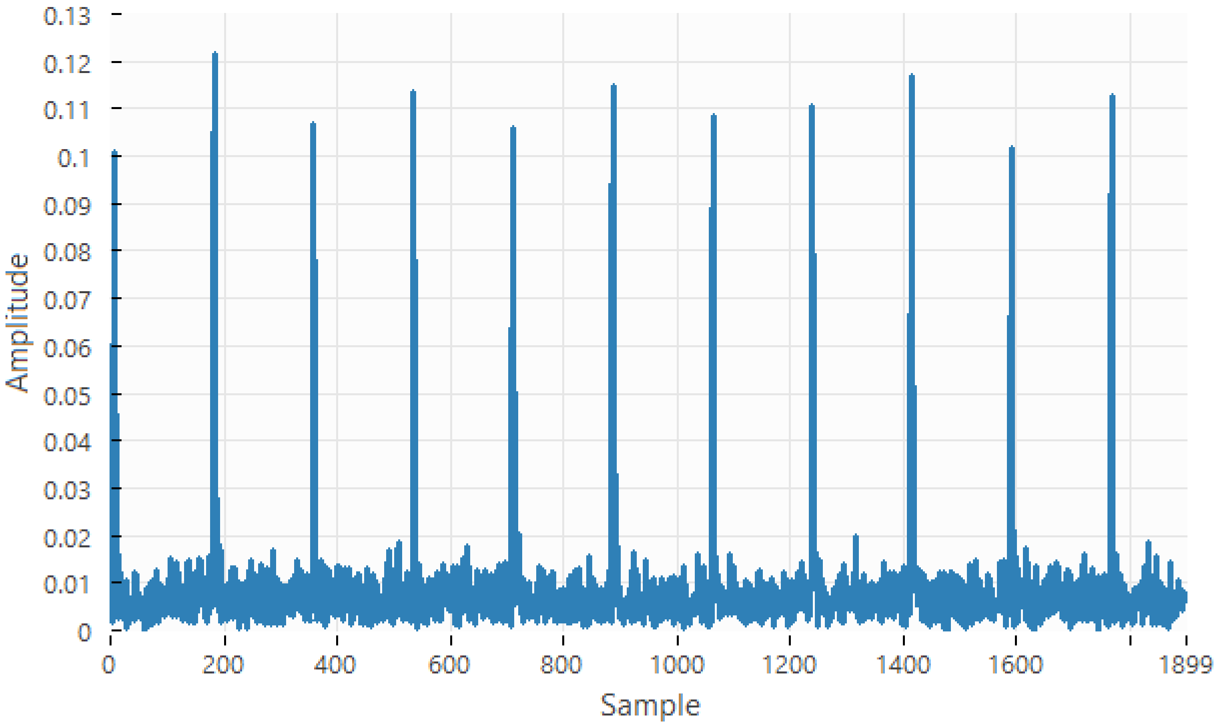
Pictures given here were obtained when calibration was performed with following parameters:

- Frequency: 74GHz
- Tx gain: -5db
- Rx gain: 31db

Immediately after generation is started Transceiver also acquires RX IQ records $Cal_{RX}(t)$.
32 records are measured, each with duration of 28000 samples (9,114µs).



Picture 3-2 Single $Cal_{RX}(t)$ *record amplitude*

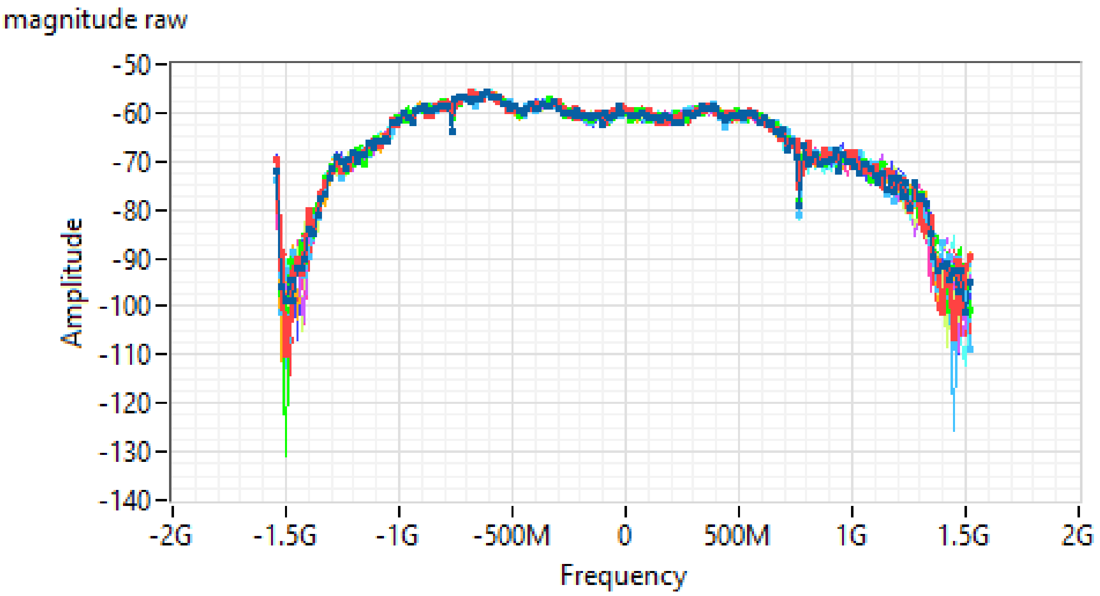


Picture 3-3 Single record Cal$_{RX}$(t) amplitude - *zoomed*

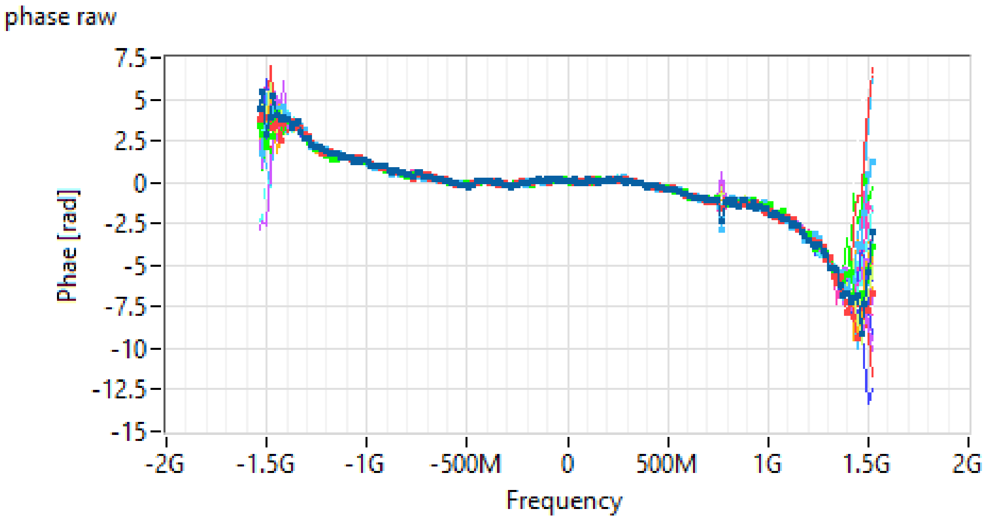
Channel frequency response *H(f)* is estimated by comparing FFT of received signal and FFT of transmitted signal in those frequency points where signal is present (since signal is periodic it has FFT only on frequencies determined by pulse repetition period)

$$H(f) = \frac{FFT\{Cal_{RX}(t)\}}{FFT\{Cal_{TX}(t)\}}$$

Channel response is calculated for each of 32 records:

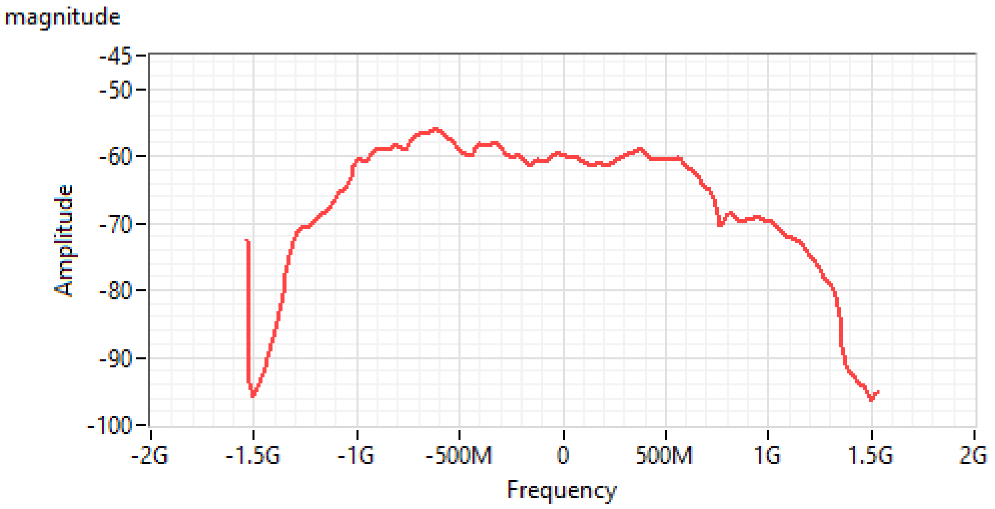


Picture 3-4 32 Channel response magnitudes for 32 records (in db)

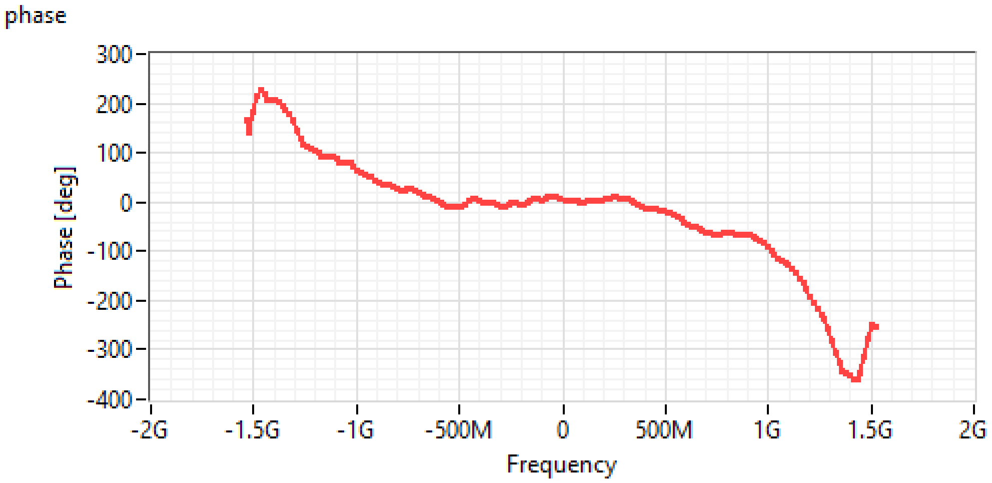


Picture 3-5 Channel response phase for 32 records (unwrapped)

Data from 32 records is averaged to obtain final channel response.

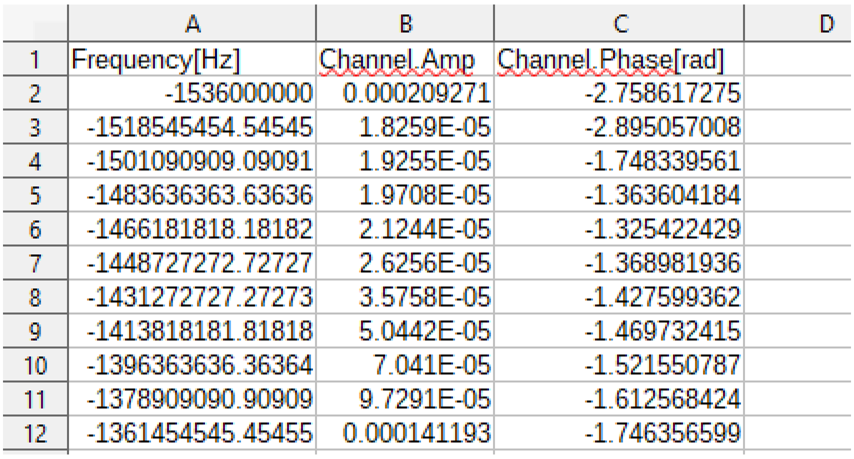


Picture 3-6 Estimated channel response magnitude



Picture 3-7 Estimate channel response phase (unwrapped in deg)

Estimated channel response will be stored in CSV file on following path:
C:/Program Data/Noffz/mmW_instr/eq_channel.csv

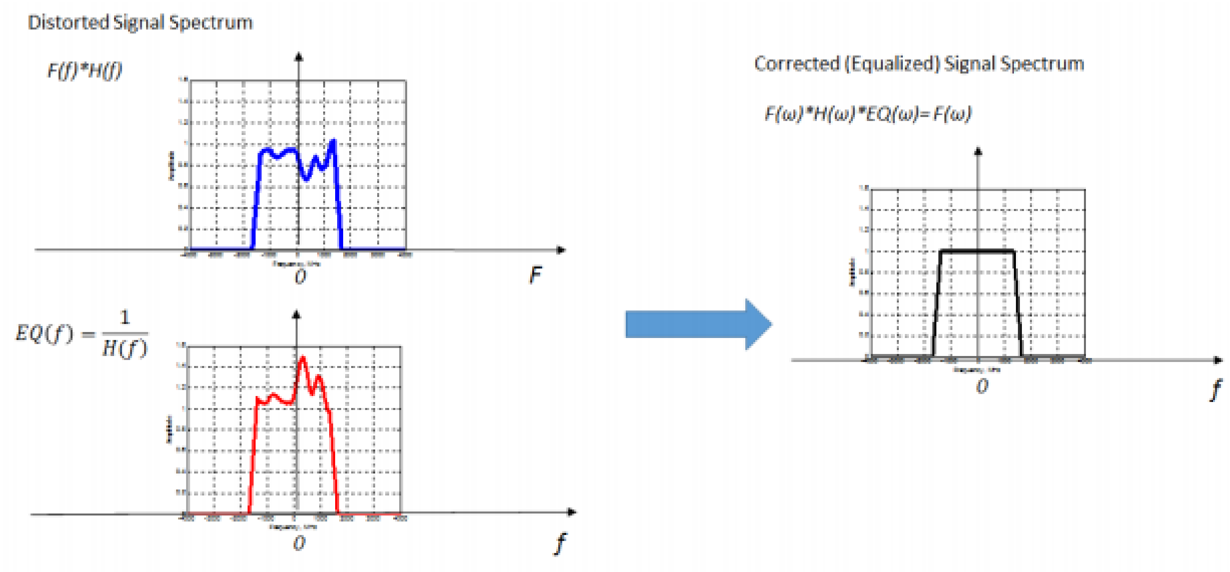


	A	B	C	D
1	Frequency[Hz]	Channel Amp	Channel Phase[rad]	
2	-1536000000	0.000209271	-2.758617275	
3	-1518545454.54545	1.8259E-05	-2.895057008	
4	-1501090909.09091	1.9255E-05	-1.748339561	
5	-1483636363.63636	1.9708E-05	-1.363604184	
6	-1466181818.18182	2.1244E-05	-1.325422429	
7	-1448727272.72727	2.6256E-05	-1.368981936	
8	-1431272727.27273	3.5758E-05	-1.427599362	
9	-1413818181.81818	5.0442E-05	-1.469732415	
10	-1396363636.36364	7.041E-05	-1.521550787	
11	-1378909090.90909	9.7291E-05	-1.612568424	
12	-1361454545.45455	0.000141193	-1.746356599	

Picture 3-8 eq_channel.csv example

3.4.1 Equalization filter calculation

Purpose of equalization filter is to flatten frequency response of system by compensating for non-flat channel response.



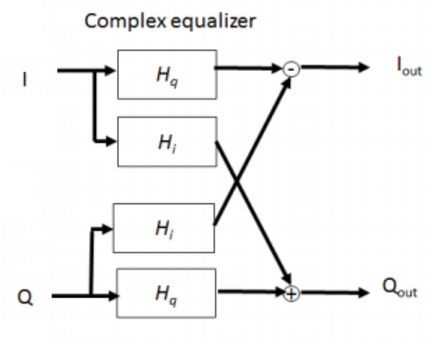
Picture 3-9 Signal spectrum correction using equalization filter

Desired response of equalization filter *EQ (f)* is calculated from normalized channel response $\overline{H}(f)$

$$\overline{H}(f) = \frac{H(f)}{\max\left(|H(f)|\right)}$$

$$EQ(f) = \frac{1}{\overline{H}(f)}$$

Since our channel response has both positive and negative frequencies (because of estimation on complex IQ signal) equalization filter also needs to be complex filter with real and imaginary path.

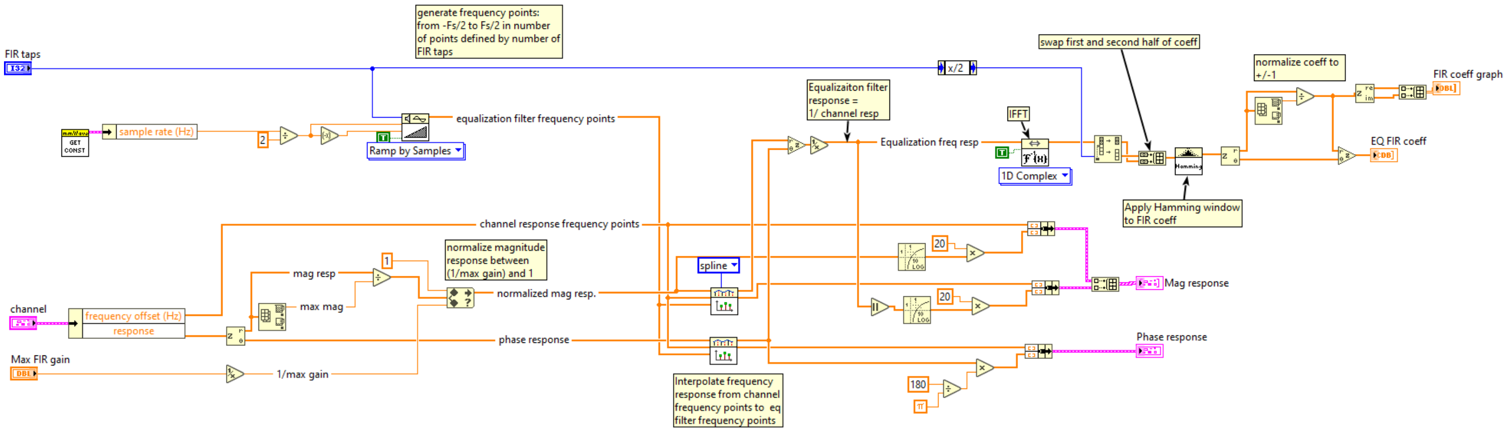


Picture 3-10 Complex equalizer structure

Equalizer will be implemented as 4 independent FIR filters. Two filters with real part of equalizer coefficients and two filters with imaginary part of filter coefficients.

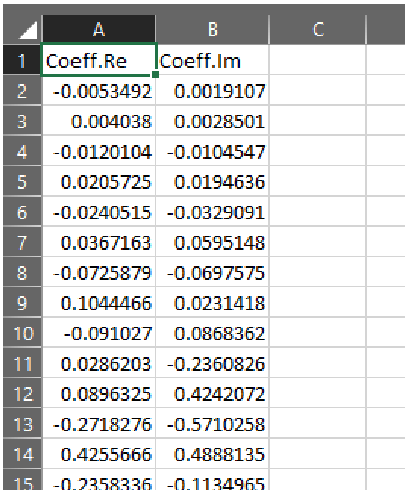
Complex equalizer coefficients are obtained from desired frequency response by filter design method which uses Inverse FFT + windowing.

On following picture LabVIEW code is given which calculates FIR coefficients for equalization filter with 32 taps.


generate frequency points: from -Fs/2 to Fs/2 in number of points defined by number of FIR taps
FIR taps
I32
sample rate (Hz)
2
Ramp by Samples
equalization filter frequency points
x/2
swap first and second half of coeff
normalize coeff to +/-1
FIR coeff graph
Equalizaiton filter response = 1/ channel resp
IFFT
Equalization freq resp
1D Complex
EQ FIR coeff
Apply Hamming window to FIR coeff
Hamming
channel response frequency points
20
normalize magnitude response between (1/max gain) and 1
mag resp
1
max mag
spline
normalized mag resp.
20
Mag response
channel
frequency offset (Hz)
response
phase response
Phase response
Max FIR gain
DBL
1/max gain
Interpolate frequency response from channel frequency points to eq filter frequency points
180
π

Picture 3-11 LabVIEW Equalization filter design suing IFFT + windowing design method

Calculated FIR coefficients are stored in CSV file:
C:/Program Data/Noffz/mmW_instr/eq_coeff.csv



	A	B	C
1	Coeff.Re	Coeff.Im	
2	-0.0053492	0.0019107	
3	0.004038	0.0028501	
4	-0.0120104	-0.0104547	
5	0.0205725	0.0194636	
6	-0.0240515	-0.0329091	
7	0.0367163	0.0595148	
8	-0.0725879	-0.0697575	
9	0.1044466	0.0231418	
10	-0.091027	0.0868362	
11	0.0286203	-0.2360826	
12	0.0896325	0.4242072	
13	-0.2718276	-0.5710258	
14	0.4255666	0.4888135	
15	-0.2358336	-0.1134965	

Picture 3-12eq_coeff.csv example

Equalization filter coefficients are downloaded to RX FPGA and they are applied in real-time to incoming signal during acquisition.

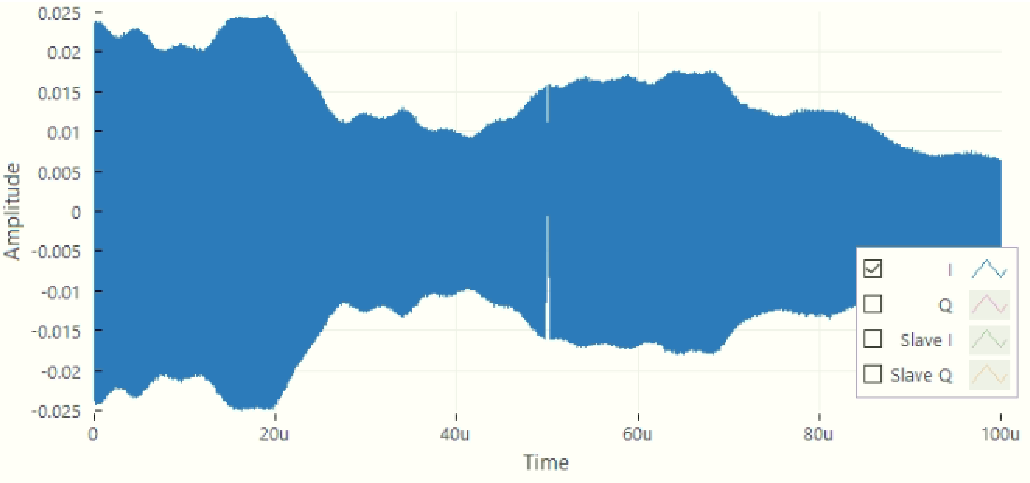
Coefficients are automatically reloaded from **eq_coeff.csv** during initialization at application start. User can change this file with custom coefficients.

3.4.2 Equalization results

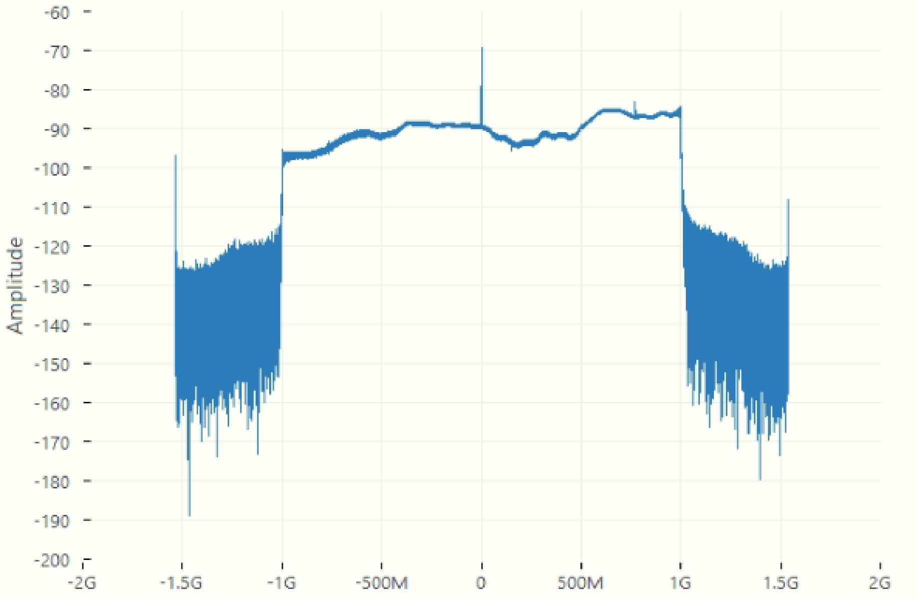
For equalization demonstration Chirp signal with 2GHz bandwidth (from -1GHz to 1GHZ) with 100 µs duration was used.

RF parameters:

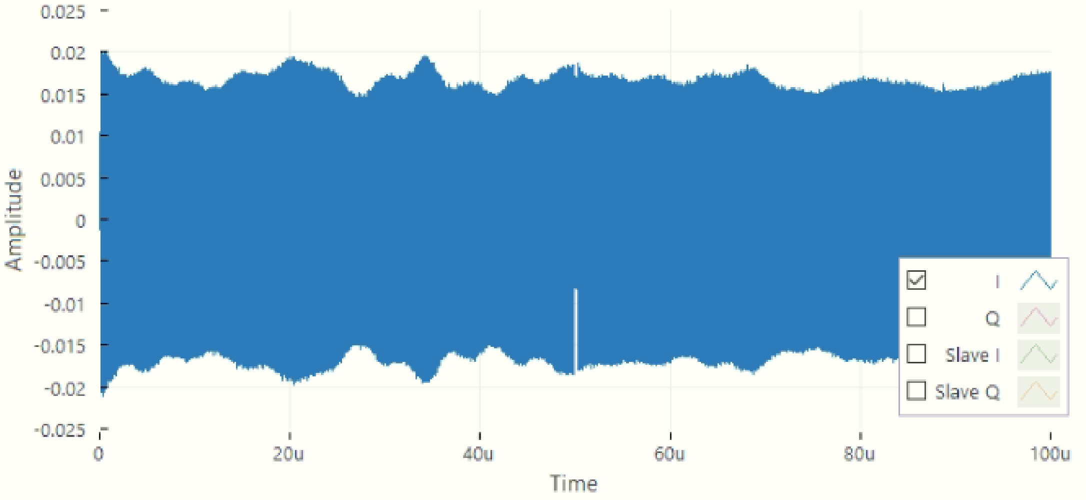
- Frequency: 74GHz
- Tx gain: -5db
- Rx gain: 31db



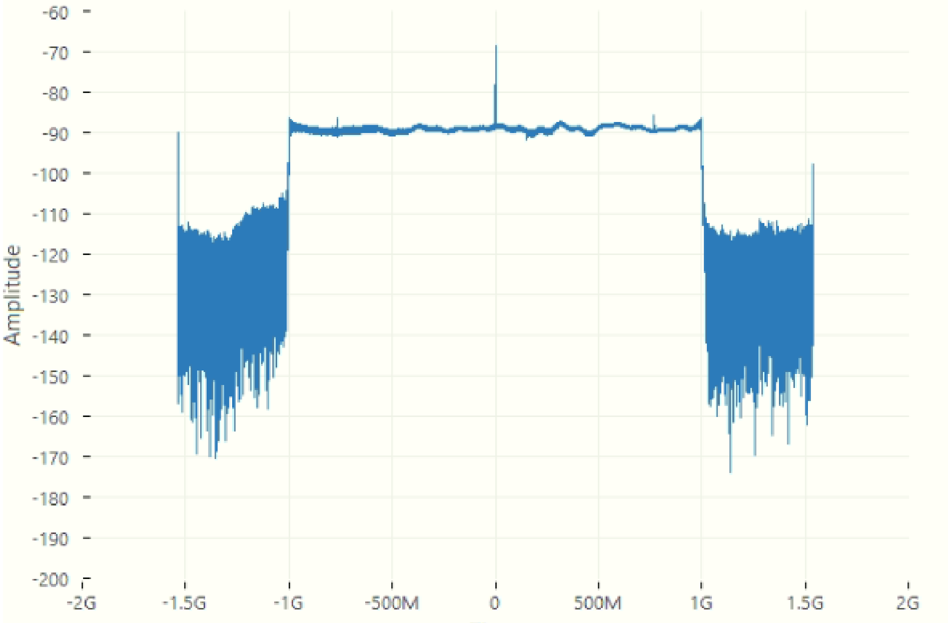
Picture 3-13 Measured RX signal of 2GHz chirp without equalization



Picture 3-14 Spectrum of measured RX signal of 2GHz chirp without equalization



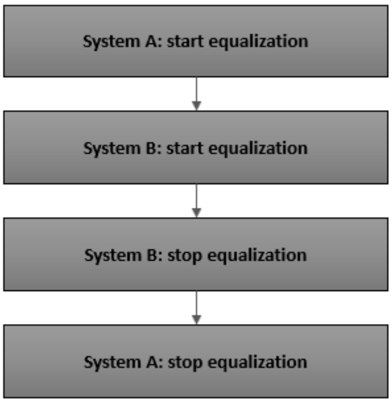
Picture 3-15 Measured RX signal of 2GHz chirp with equalization



Picture 3-16 Spectrum of measured RX signal of 2GHz chirp with equalization

3.4.3 Multi-system equalization

Equalization process for both systems is given on following chart:


System A: start equalization
System B: start equalization
System B: stop equalization
System A: stop equalization

Picture 3-17 Equalization flow

System A should enter Calibration state before System B.
While System A is in Calibration state it will continue generating Calibration signal even after it completes calculation of its own equalization coefficients.
System B will use calibration signal from system A to estimate channel response between System A DAC and System B ADC.

3.5 TX signal generation

TX IQ signal is generated using Tx_FPGA and Tx_DAC modules.

System can generate arbitrary IQ samples with configurable sample frequency and arbitrary duration.

Samples are provided in complex form via Write_tx command through API or in **.dat** file (when using local GUI). Samples are provided in form of "bursts", each burst has name and array of complex samples.

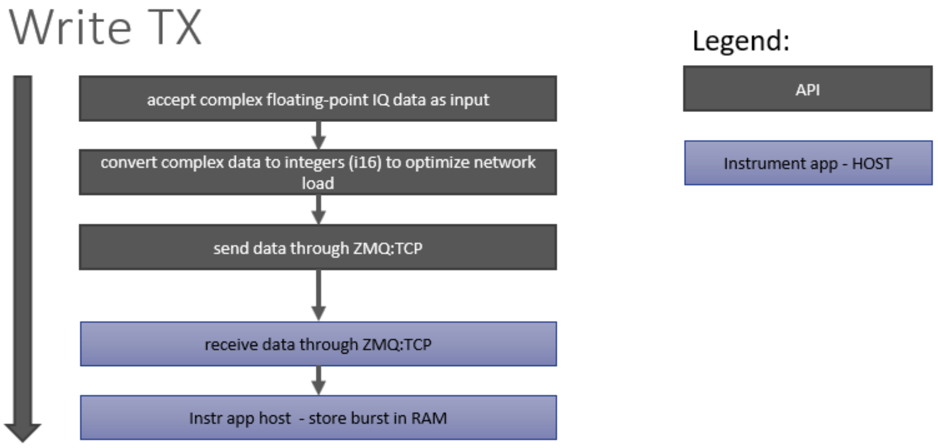
Following generation features are supported:

- Generate single burst
- Generate burst continuously (repeats burst generation until new burst is generated or stop command is issued)
- Generate burst "playlist": Several bursts are generated one after another. Bursts can repeat inside playlist without using additional memory. (only when using low level command)
- Generate burst "playlist" continuously.

Once bursts generation is started (System is in **Running**) state burst or playlist can be retriggered arbitrarily without providing new samples.

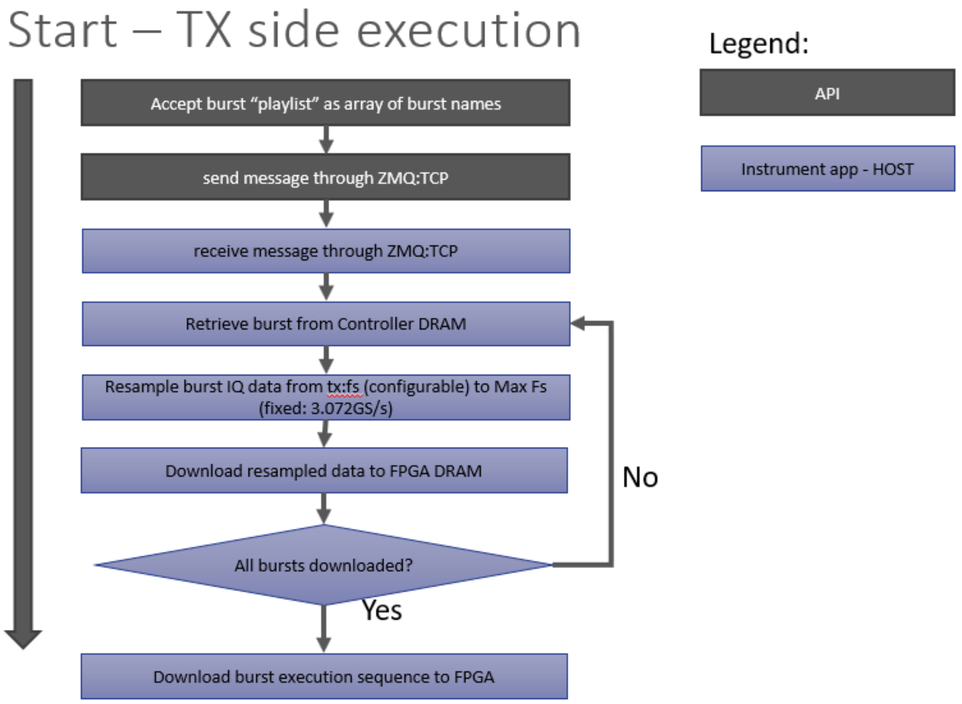
3.5.1 Using low level commands

- With **Write TX** command IQ data is provided as array and memorized on Controller under given name
- If **Write TX** command is given with burst name of existing burst, its content will be overwritten in memory.


Write TX
accept complex floating-point IQ data as input
convert complex data to integers (i16) to optimize network load
send data through ZMQ:TCP
receive data through ZMQ:TCP
Instr app host - store burst in RAM
Legend:
API
Instrument app - HOST

Picture 3-18 Write TX command execution

- **Start** command provides burst "playlist": order of bursts to be generated.
- When **Start** command is received bursts are retrieved from Controller RAM memory, resampled from tx fs to maximum sampling frequency and downloaded into FPGA through DMA. On FPGA samples are bit-packed into DRAM memory. Bursts playlist is then downloaded to FPGA so bursts generation can be sequenced in given order.


Start – TX side execution
Accept burst "playlist" as array of burst names
send message through ZMQ:TCP
receive message through ZMQ:TCP
Retrieve burst from Controller DRAM
Resample burst IQ data from tx:fs (configurable) to Max Fs (fixed: 3.072GS/s)
Download resampled data to FPGA DRAM
All bursts downloaded?
No
Yes
Download burst execution sequence to FPGA
Legend:
API
Instrument app - HOST

Picture 3-19 Start command execution - (TX side)

- When **Send trigger** command is received generation and acquisition start synchronously (on one or both systems depending on trigger sharing).Tx_FPGA starts reading burst samples from its DRAM memory in order given by playlist one after another. If generation mode is set to continuous once all bursts from playlist are generated generation restarts from first burst.
- If generation mode is continuous burst sequence will be continuously generated until **stop** command is sent

3.5.2 TX FPGA: DRAM memory bit-packing and signal length

Tx_FPGA is PXIe-7902 module with large FPGA chip and DRAM memory which is external to it.

External DRAM has 2GB available organized in 33554432 addresses which hold 512 bits.

For DRAM to operate at its max speed (10.5 GB/s) following must be met:

- all bits from one address must be used
- addresses should be written/read in succession.
- DRAM should only be used for write OR read operations at a time.

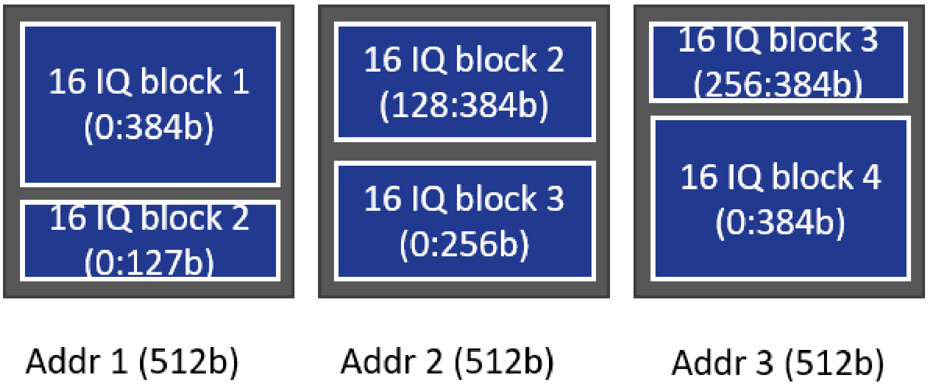
DAC module output data with 14bit precision at end but interface is 16bit.

To generate long burst properly we need to provide DAC module through its digital interface (MGT interface) new samples at 3.072GS/s rate. On FPGA side this is handled as sending 16 by 16 IQ samples (real + imaginary) at each clock tick of 192MHz clock. Data required is 16-bit precision for I and Q (4Bytes for whole complex sample).

This data throughput is 192M x 16 x 4B = 12.288 GB/s which is higher than maximum DRAM throughput.

Therefore, data is stored in DRAM with 12-bit precision in following way:

4 "blocks" of 16 12-bit IQ samples are stored in 3 DRAM addresses (4 x 16 x (12b + 12b) = 4 x 384b = 1536b = 3 x 512b)



Picture 3-20 DRAM bit-packing. Storing 64 12-bit precision IQ samples into 3 DRAM addresses

This reduces data throughput requested from DRAM to 9.216GB/s which is withing its capability. Data is converted from 12 to 16-bit precision prior to sending to DAC.

Note: Because data is bit-packet burst length resolution for both TX and RX is 4*16 = 64Samples. When burst is specified with length different from multiple of 64 it will be zero padded until correct length. This needs to be considered when generating in continuous mode.

3.6 RX signal acquisition

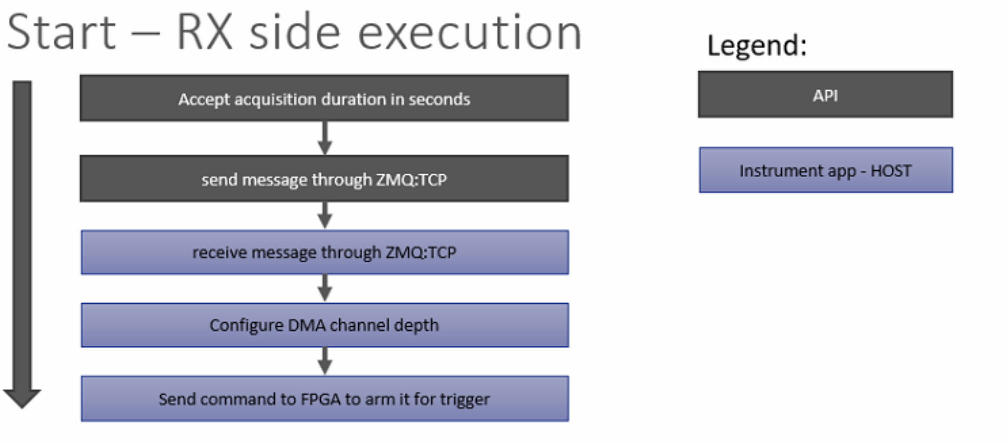
RX IQ samples are acquired using Rx_ADC and Rx_FPGA modules.

System can acquire IQ samples with arbitrary number of samples.
Samples can be retrieved from system using **Fetch** command. Samples can be retrieved via API or via local GUI. Samples are provided in form of “records”, each “record” has name and array of complex samples.

Once system is in **Running** state record will be acquired on each trigger.

3.6.1 Using low level commands

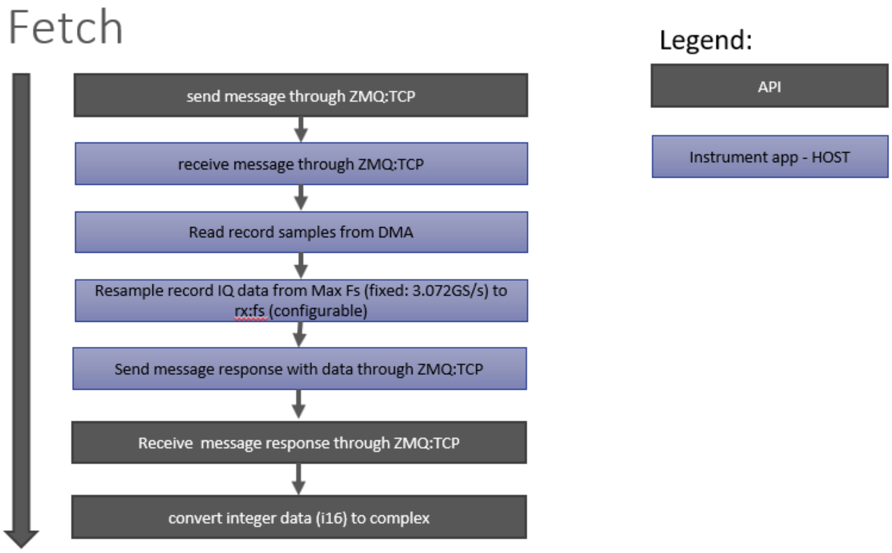
- **Start** command provides acquisition record length in seconds. This duration will be converted in number of samples at max fs (3.072GS/s).
- When **Start** command is received system goes into **Running** state and is prepared for receiving triggers.


Start – RX side execution
Accept acquisition duration in seconds
send message through ZMQ:TCP
receive message through ZMQ:TCP
Configure DMA channel depth
Send command to FPGA to arm it for trigger
Legend:
API
Instrument app - HOST

Picture 3-21 Start command execution (RX side)

- When **Send trigger** command is received generation and acquisition start synchronously (on one or both systems depending on trigger sharing). Rx_FPGA starts reading IQ samples from ADC, bit-packs them and writes them into DRAM until number all samples of record are stored. Once record storing is done FPGA will start reading data from DRAM and forwarding them through DMA to Controller.

- **Fetch** command retrieves complete record from FPGA DMA channel. Data is then resampled from maximum Fs (3.072GS/s) to configurable rx fs. If command was issued through API samples are sent back to API caller through communication. For local UI and debugging purposes last several records are kept in Controller RAM memory (up to 500MB of record data) for visualization and processing in GUI.


Fetch
send message through ZMQ:TCP
receive message through ZMQ:TCP
Read record samples from DMA
Resample record IQ data from Max Fs (fixed: 3.072GS/s) to rx:fs (configurable)
Send message response with data through ZMQ:TCP
Receive message response through ZMQ:TCP
convert integer data (i16) to complex
Legend:
API
Instrument app - HOST

Picture 3-22 Fetch command

3.6.2 RX FPGA: DRAM memory bit-packing

IQ samples for RX are bit-packed into onboard DRAM memory on Rx_FPGA in same way as TX bursts. DRAM is used as one big buffer for RX data.

Maximum number of samples which can be buffered is 33554432 x (4/3) x 16 = 715,827,882 samples which is equivalent to 233ms of data.

4 Software

Software consists of 3 modules:

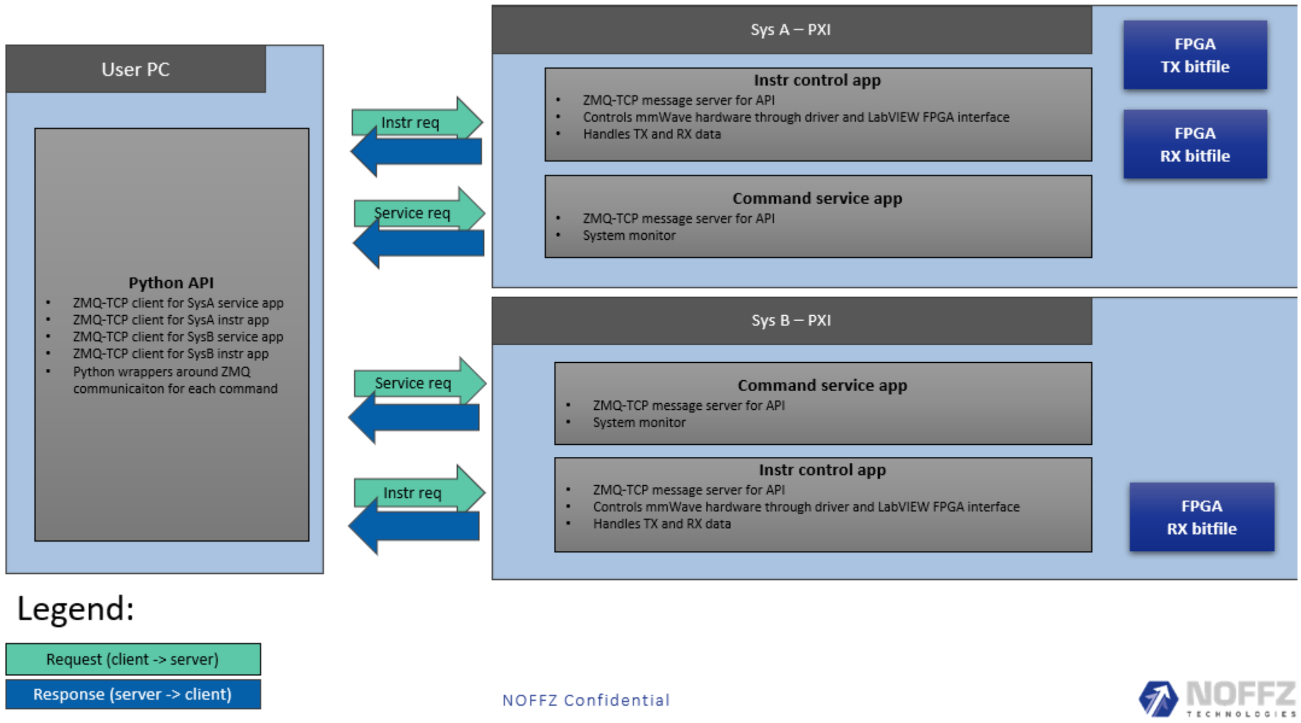
- mmWave instrument control app
- Command service app
- Python API

Instrument control and service app are implemented in LabVIEW 2019 (64bit) while Python API was implemented using Python 3.7 (64bit).

Modules and apps are communicating with each other using ZeroMQ (ZMQ) TCP library:

- ZMQ is lightweight protocol based on TCP which provides API for multiple different targets and programming languages
- It provides connection management
- ZMQ has several communication patterns, in this application we are using Request – Response
- ZMQ manages connection and communication in background asynchronous IO thread.

LabVIEW FPGA module was used to extend reference FPGA code for digital baseband for both TX and RX.


User PC
Python API
ZMQ-TCP client for SysA service app
ZMQ-TCP client for SysA instr app
ZMQ-TCP client for SysB service app
ZMQ-TCP client for SysB instr app
Python wrappers around ZMQ communicaiton for each command
Instr req
Service req
Service req
Instr req
Sys A – PXI
Instr control app
ZMQ-TCP message server for API
Controls mmWave hardware through driver and LabVIEW FPGA interface
Handles TX and RX data
Command service app
ZMQ-TCP message server for API
System monitor
FPGA TX bitfile
FPGA RX bitfile
Sys B – PXI
Command service app
ZMQ-TCP message server for API
System monitor
Instr control app
ZMQ-TCP message server for API
Controls mmWave hardware through driver and LabVIEW FPGA interface
Handles TX and RX data
FPGA RX bitfile
Legend:
Request (client -> server)
Response (server -> client)
NOFFZ Confidential
NOFFZ TECHNOLOGIES

Picture 4-1 Software architecture

4.1 Instrument control app

Instrument control application is developed using LabVIEW 2019 and LabVIEW 2019 FPGA.

Application is installed on following path:

C:\Program Files\Noffz\Barkhausen mmWave radar

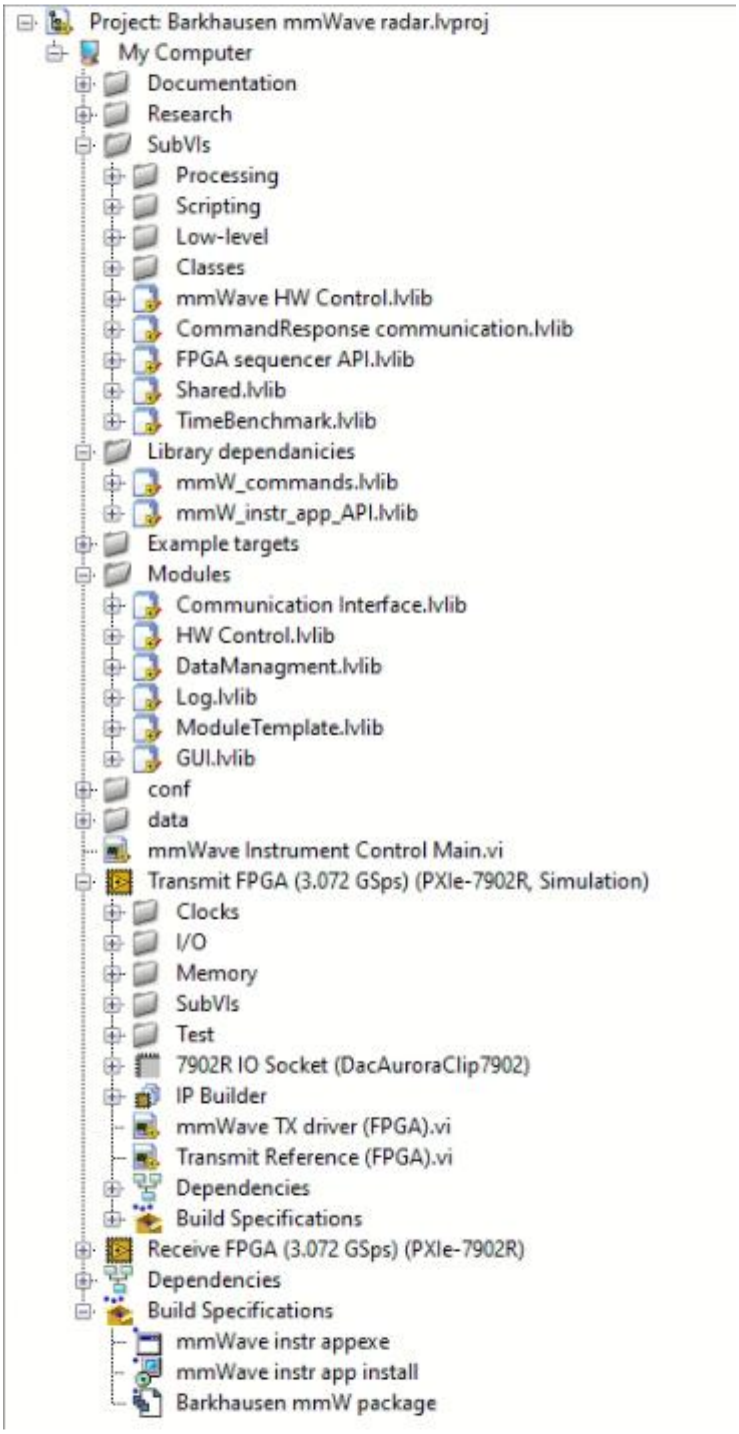
Configuration files for application are located on following path:

C:\ProgramData\Noffz\mmW_instr

Purpose of application is to provide link between user commands and hardware.

Complete application with FPGA codes is organized in single LabVIEW project with following structure

- **My Computer -** Windows app
  - **Documentation –** initial LV project doc
  - **Research –** various LV Vis used for POC and experiments
  - **SubVIs –**VIs, libraries and classes used in code
  - **Library Dependencies –** external libraries (from other modules)
  - **Example targets –** starting project template targets
  - **Modules –** libraries for all instr app modules
  - **conf –** configuration files (default values)
  - **data –** ZMQ and other dll dependencies
  - **mmWave Instrument Control Main.vi – main VI**
- **Transmit FPGA (3.072 GSps) –** Tx_FPGA code
- **Receive FPGA (3.072 GSps) –** Rx_FPGA code
- **Build specifications** – distributaion generation



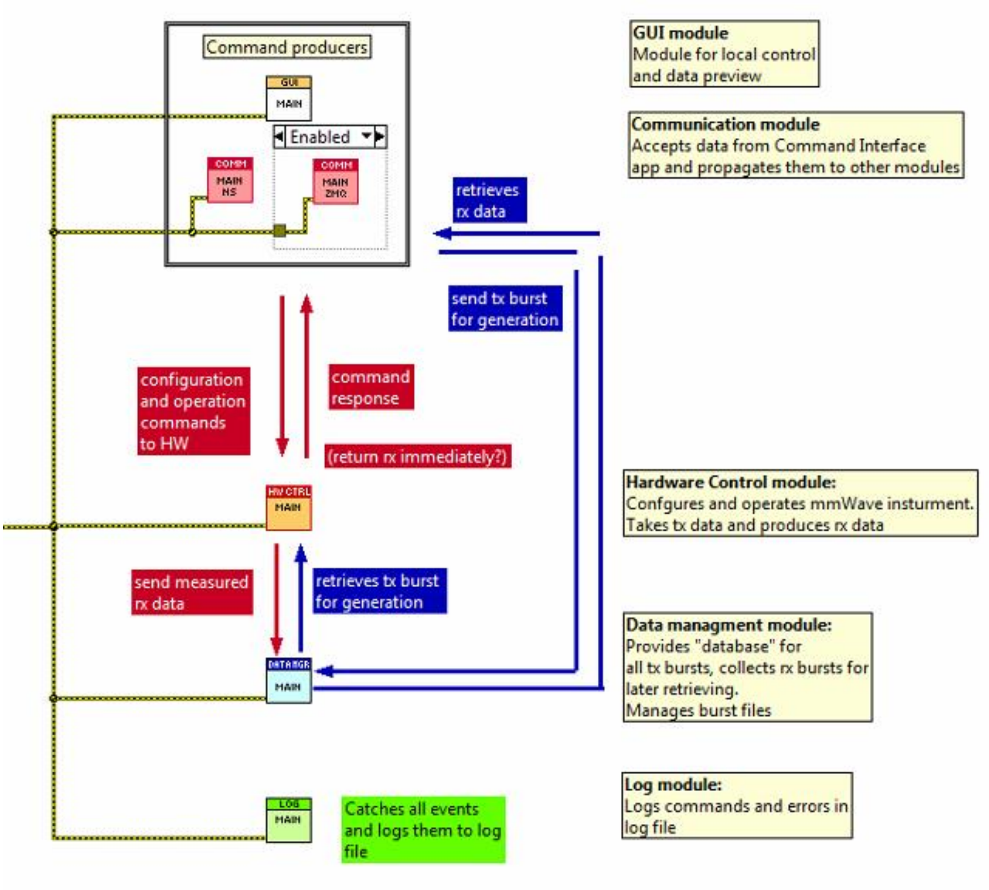
- Project: Barkhausen mmWave radar.lvproj
  - My Computer
    - Documentation
    - Research
    - SubVIs
      - Processing
      - Scripting
      - Low-level
      - Classes
      - mmWave HW Control.lvlib
      - CommandResponse communication.lvlib
      - FPGA sequencer API.lvlib
      - Shared.lvlib
      - TimeBenchmark.lvlib
    - Library dependanicies
      - mmW_commands.lvlib
      - mmW_instr_app_API.lvlib
    - Example targets
    - Modules
      - Communication Interface.lvlib
      - HW Control.lvlib
      - DataManagment.lvlib
      - Log.lvlib
      - ModuleTemplate.lvlib
      - GUI.lvlib
    - conf
    - data
    - mmWave Instrument Control Main.vi
    - Transmit FPGA (3.072 GSps) (PXIe-7902R, Simulation)
      - Clocks
      - I/O
      - Memory
      - SubVIs
      - Test
      - 7902R IO Socket (DacAuroraClip7902)
      - IP Builder
      - mmWave TX driver (FPGA).vi
      - Transmit Reference (FPGA).vi
      - Dependencies
      - Build Specifications
    - Receive FPGA (3.072 GSps) (PXIe-7902R)
    - Dependencies
    - Build Specifications
      - mmWave instr appexe
      - mmWave instr app install
      - Barkhausen mmW package

Picture 4-2 Instrument app LabVIEW project

Application (mmWave Instrument Control Main.vi) consists of several concurrent processes:

- **Hardware Control module:** Accepts commands and controls hardware via *Hardware Abstraction Layers*.
- **Communication module(s):** One module receives commands remotely using Network Streaming (Debug/development) and other one ZMQ (API). Commands are passed to Hardware Control module
- **GUI:** Local user interface which sends commands to Hardware Control Module and plots TX and RX IQ waveforms

- **Data management module:** Keeps and manages preloaded TX waveforms and retrieved RX waveforms in memory
- **Log module:** Logs all received commands with their parameters and errors which occurred in code.


Command producers
GUI
MAIN
Enabled
COMM
MAIN
NS
COMM
MAIN
ZMQ
retrieves rx data
send tx burst for generation
configuration and operation commands to HW
command response
(return rx immediately?)
HW CTRL
MAIN
send measured rx data
retrieves tx burst for generation
DATAMGR
MAIN
LOG
MAIN
Catches all events and logs them to log file
GUI module
Module for local control and data preview
Communication module
Accepts data from Command Interface app and propagates them to other modules
Hardware Control module:
Confgures and operates mmWave insturment.
Takes tx data and produces rx data
Data managment module:
Provides "database" for all tx bursts, collects rx bursts for later retrieving.
Manages burst files
Log module:
Logs commands and errors in log file

Picture 4-3 Instrument control app block diagram

Inter-module communication is implemented using LabVIEW user events.
User events provide many-to-many communication mechanism. Event can be generated on several different places modules and registered (handled) wherever needed.
Each event provides message (string), data (variant/anything) and notifier for response.

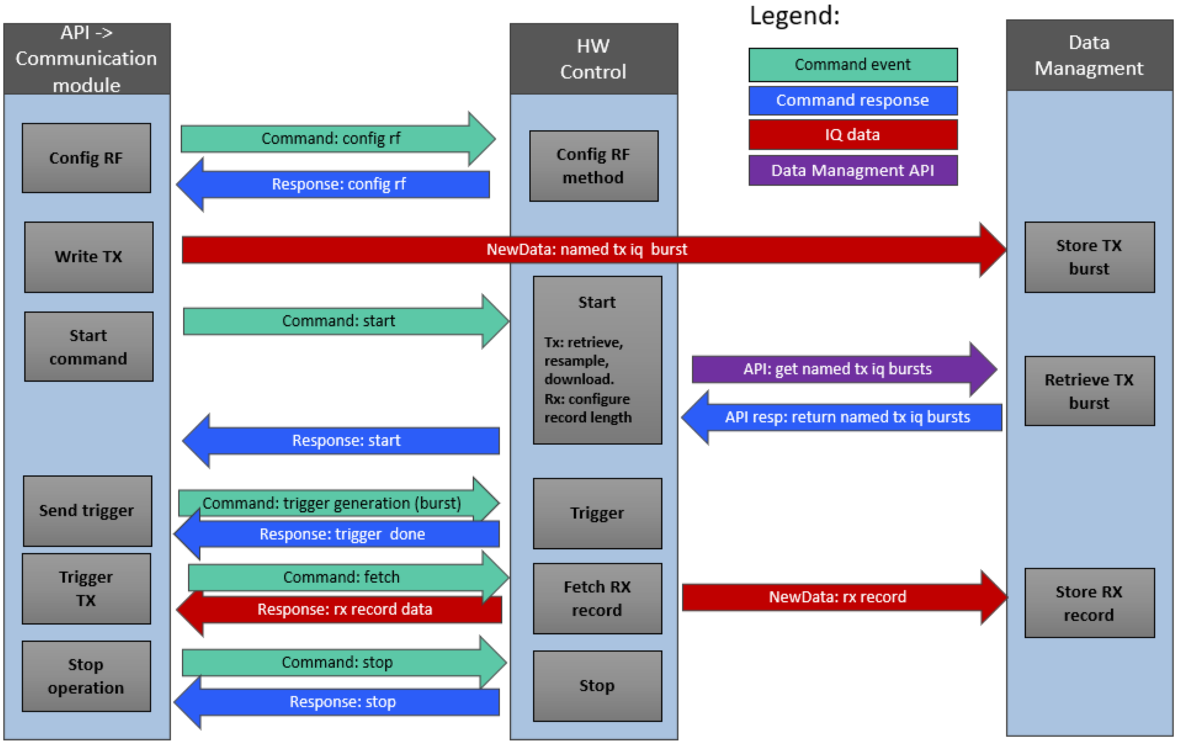
Following events are present in application:

- **Command:** New command was issued to instrument. Generated by communication module (accepting API command) or GUI
- **NewData:** New IQ samples available. TX bursts: generated by GUI and Communication module when new IQ samples are present. RX records: generated by HW Control module when data is fetched
- **CloseApp:** Event generated when application is closing.
- **Error:** Generated by any module when LabVIEW error occurs

Following module have “internal” user events used for self-signaling and wrapped inside API functions for those modules:

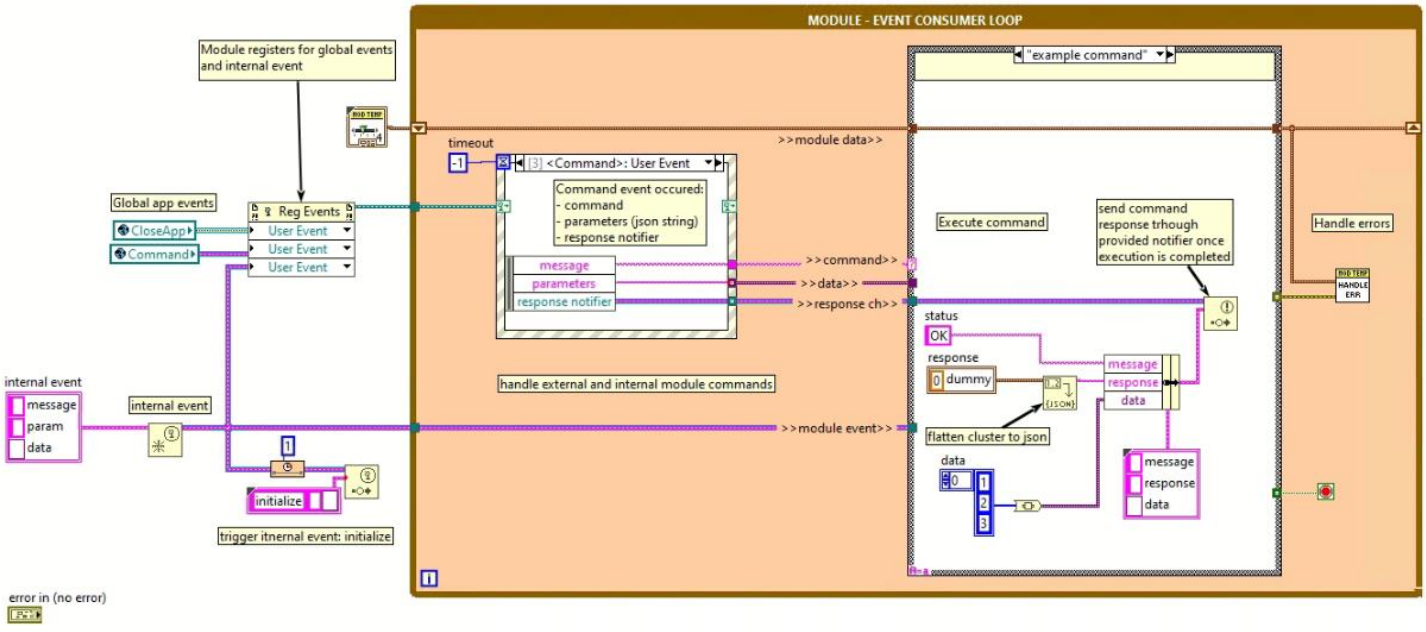
- **Log event:** Send custom log message to Log module
- **DM event:** Event which provides interface to and from Data management module

Following chart illustrates example event flow:
Communication module accepts incoming API commands, processes them and triggers **Command** event or **NewData** event. **HW Control** is registered to **Command** event to which it provides **Responses** upon completing hardware actions. It uses **DataManagment** API to retrieve bursts and **NewData** to signal new RX records are present in system. **DataManagment** module register when new data is available. It also provides data back when requested through its API.



Picture 4-4 Example evnt flow during TX/RX operations

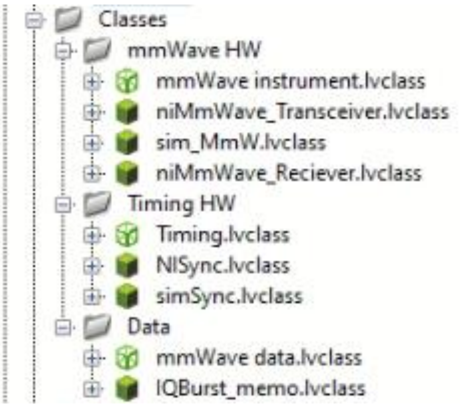
Modules generally are implemented as Event consumers with following template:



Picture 4-5 Instrument application module template

4.1.1 Classes

Project contains several classes for various purposes.
All classes were created using AddQ G# framework (http://sine.ni.com/nips/cds/view/p/lang/en/nid/209103) .
G# extends basic LabVIEW OOP to memory optimal reference-based OOP and provides various extensions to IDE which speed up class development.



```
Classes
    mmWave HW
        mmWave instrument.lvclass
        niMmWave_Transceiver.lvclass
        sim_MmW.lvclass
        niMmWave_Reciever.lvclass
    Timing HW
        Timing.lvclass
        NISync.lvclass
        simSync.lvclass
    Data
        mmWave data.lvclass
        IQBurst_memo.lvclass
```

Picture 4-6 mmW project classes

There are 3 groups of classes:

- **mmWave HW:** Hardware abstraction layer classes for mmWave instrumentation
- **Timing HW:** Hardware abstraction layer classes
- **Data:** Classes for reference memory optimal burst/record IQ data management

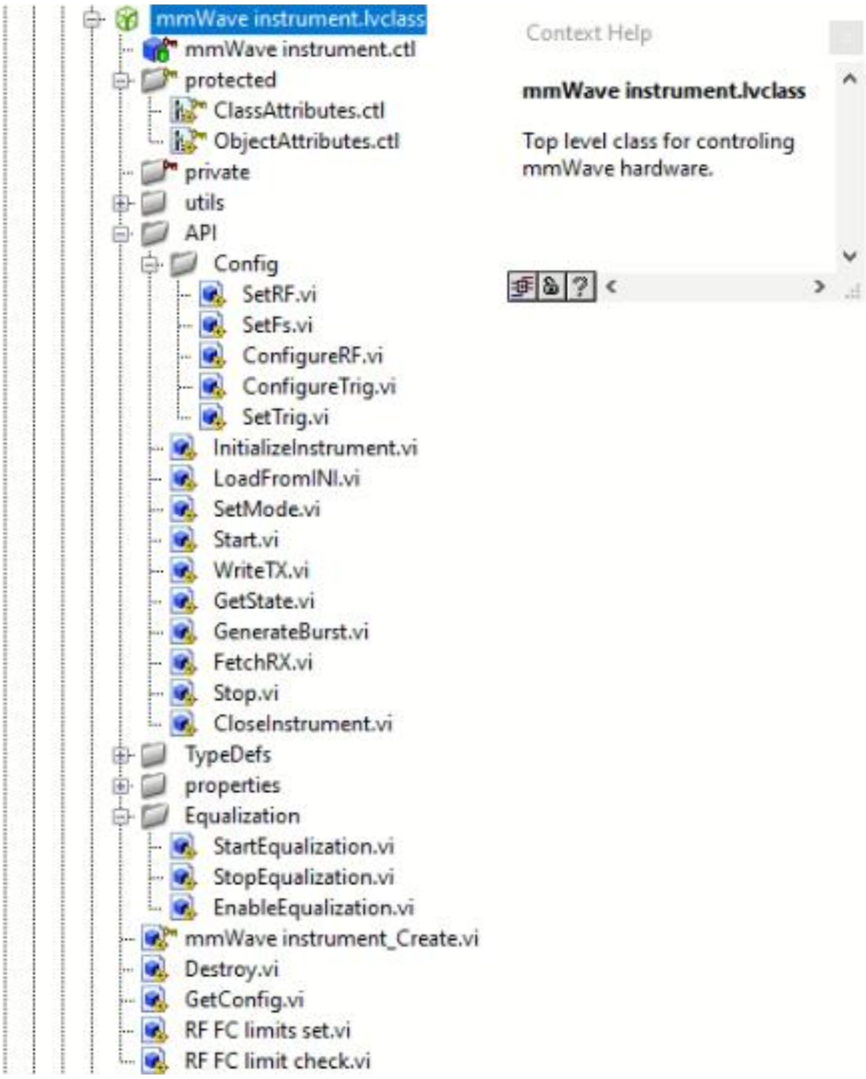
Hardware abstraction layer provides interface between hardware usage and driver calls. They provide methods with clear and simple API and in their implementation API inputs are adapted to driver calls. HALs are created in a way that there is always top-level *abstract* class which provides API interface. From this top-level parent class child classes are derived for specific instruments. All other modules which use hardware use it through parent-class methods.

This implementation allows dependency-injection (changing instrument used by just changing which child class is used without need to modify module code). For example, HW Control module without any additional changes can handle SysA hardware, SysB hardware and simulated hardware.

4.1.1.1 mmWave HAL

Parent abstract class: mmWave instrument.lvclass

Provides attributes shared by all child classes. Provides methods to be overwritten



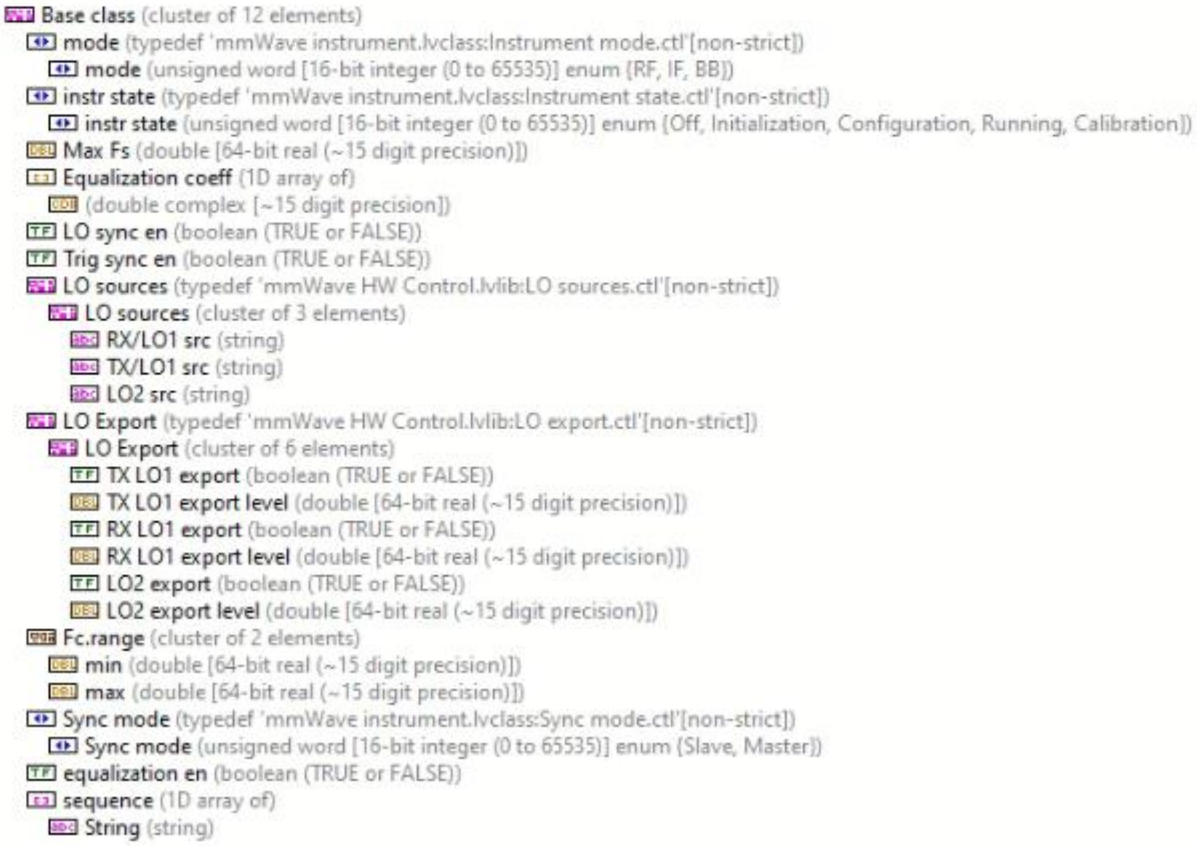
```
mmWave instrument.lvclass
    mmWave instrument.ctl
    protected
        ClassAttributes.ctl
        ObjectAttributes.ctl
    private
    utils
    API
        Config
            SetRF.vi
            SetFs.vi
            ConfigureRF.vi
            ConfigureTrig.vi
            SetTrig.vi
        InitializeInstrument.vi
        LoadFromINI.vi
        SetMode.vi
        Start.vi
        WriteTX.vi
        GetState.vi
        GenerateBurst.vi
        FetchRX.vi
        Stop.vi
        CloseInstrument.vi
    TypeDefs
    properties
    Equalization
        StartEqualization.vi
        StopEqualization.vi
        EnableEqualization.vi
    mmWave instrument_Create.vi
    Destroy.vi
    GetConfig.vi
    RF FC limits set.vi
    RF FC limit check.vi
```

Context Help

mmWave instrument.lvclass

Top level class for controling mmWave hardware.

Picture 4-7 Abstract top level class for mmW instrument control



```
Base class (cluster of 12 elements)
  mode (typedef 'mmWave instrument.lvclass:Instrument mode.ctl'[non-strict])
    mode (unsigned word [16-bit integer (0 to 65535)] enum {RF, IF, BB})
  instr state (typedef 'mmWave instrument.lvclass:Instrument state.ctl'[non-strict])
    instr state (unsigned word [16-bit integer (0 to 65535)] enum {Off, Initialization, Configuration, Running, Calibration})
  Max Fs (double [64-bit real (~15 digit precision)])
  Equalization coeff (1D array of)
    (double complex [~15 digit precision])
  LO sync en (boolean (TRUE or FALSE))
  Trig sync en (boolean (TRUE or FALSE))
  LO sources (typedef 'mmWave HW Control.lvlib:LO sources.ctl'[non-strict])
    LO sources (cluster of 3 elements)
      RX/LO1 src (string)
      TX/LO1 src (string)
      LO2 src (string)
  LO Export (typedef 'mmWave HW Control.lvlib:LO export.ctl'[non-strict])
    LO Export (cluster of 6 elements)
      TX LO1 export (boolean (TRUE or FALSE))
      TX LO1 export level (double [64-bit real (~15 digit precision)])
      RX LO1 export (boolean (TRUE or FALSE))
      RX LO1 export level (double [64-bit real (~15 digit precision)])
      LO2 export (boolean (TRUE or FALSE))
      LO2 export level (double [64-bit real (~15 digit precision)])
  Fc.range (cluster of 2 elements)
    min (double [64-bit real (~15 digit precision)])
    max (double [64-bit real (~15 digit precision)])
  Sync mode (typedef 'mmWave instrument.lvclass:Sync mode.ctl'[non-strict])
    Sync mode (unsigned word [16-bit integer (0 to 65535)] enum {Slave, Master})
  equalization en (boolean (TRUE or FALSE))
  sequence (1D array of)
    String (string)
```

Picture 4-8 Base class attributes

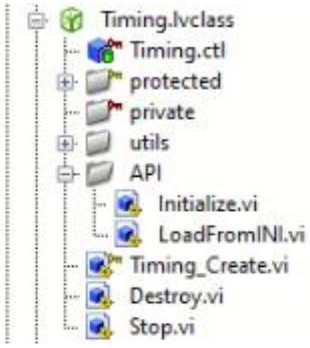
From this class 3 child classes are derived:

- **niMmWave_Transciever.lvclass:** HAL for controlling System A hardware. Supports TX generation and RX acquisition. Provides interface to all mmW hardware (FPGAs, ADC, DAC, IF)
- **niMmWave_reciever.lvclass:** HAL for controlling System B hardware. Supports RX acquisition. Provides interface to RX mmW hardware (FPGA, ADC, IF)
- **sim_MmW.lvclass:** Simulated HAL. Can be used on PC without hardware. Supports TX generation and TX acquisition. Provides simulated signal response based on "generated" TX burst.

4.1.1.2 Timing HW HAL

Parent abstract class: Timing.lvclass
Class provides two main methods and no shared attributes:

- LoadFromINI.vi – loads object data from ini file
- Initialize.vi – Makes all trigger connections (exporting/importing)



```
Timing.lvclass
  Timing.ctl
  protected
  private
  utils
  API
    Initialize.vi
    LoadFromINI.vi
  Timing_Create.vi
  Destroy.vi
  Stop.vi
```

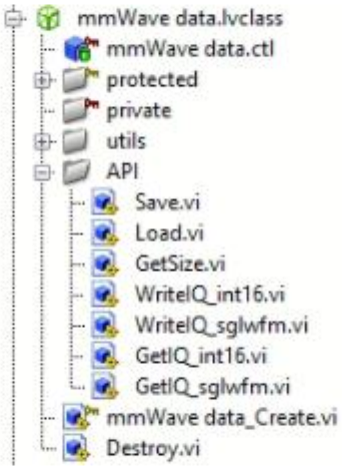
Picture 4-9 Abstract top level class for timing module control

4.1.1.3 mmWave data

mmWave data classes provide wrappers around IQ data. Since G# objects are reference based this enables data sharing between modules without additional data copies. Sending bursts and records between methods is implemented as sending G# objects which are lightweight (they contain only pointer to object attributes where data is held).

There is abstact class: **mmWave data.lvclass** from which single child class is derived **IQBurst_memo.lvclass**

mmWave data.lvclass provides methods for writing and reading data in various formats, saving, loading and retrieving data size in Bytes.
IQBurst_memo.lvclass keeps data in RAM memory as array on int16 data, I and Q samples interleaved.



```
mmWave data.lvclass
    mmWave data.ctl
    protected
    private
    utils
    API
        Save.vi
        Load.vi
        GetSize.vi
        WriteIQ_int16.vi
        WriteIQ_sglwfm.vi
        GetIQ_int16.vi
        GetIQ_sglwfm.vi
    mmWave data_Create.vi
    Destroy.vi
```

Picture 4-10 mmWave data class

4.1.2 Hardware Control Module

Configures and operates mmWave and Timing module instruments. Takes tx data and produces rx data.

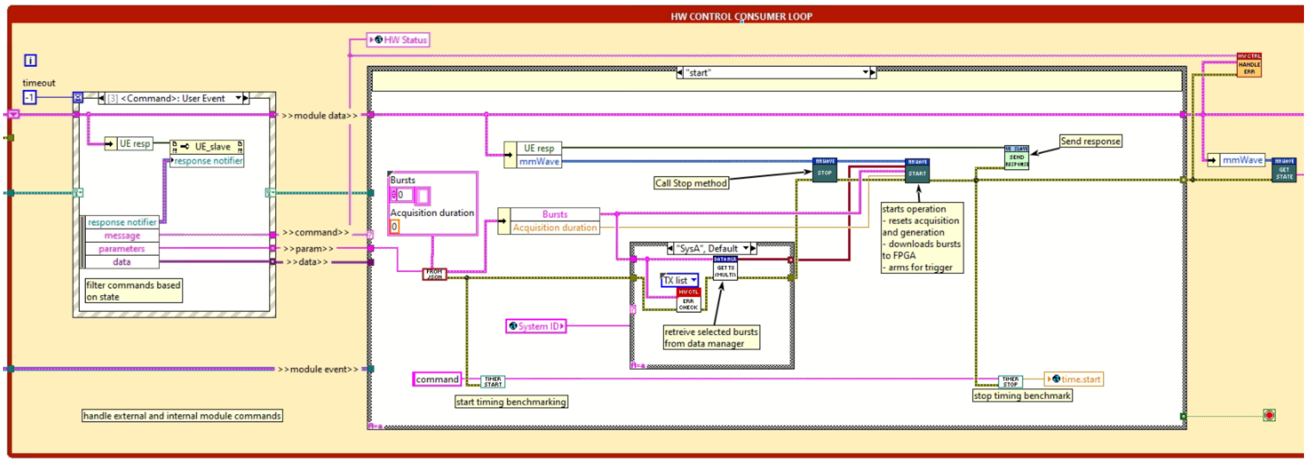
Hardware is controlled via HAL Classes for mmW and Timing module.

At start: module creates all HAL objects based on ini file and initializes them in default mode (specified by ini)

Module is registered for following events:

- **Command**
- **CloseApp**
- **HW module internal event**

Module processes received commands by calling HAL methods and returns response.
In case error occurs module handles it by forcing HAL in safe state: Configuration



Picture 4-11Start command handling in HW Control module

4.1.3 Communication module

Provides entry point for commands received through ZMQ.

Module is registered for following event:

- **CloseApp**

Note: Instrument Application includes the “mmW Library” and uses its VIs to construct and extract messages to send and receive. For detailed description of the defined message formats please refer to the the “mmw Library” documentation, chapter “Communication channel and Data Types”

Here is a short summary of the massage formats:

Open ZMQ:REP socket and waits for messages.
When message is received following data is extracted:

- Message: string
- Parameters (optional): JSON formatted string, extracted as LabVIEW cluster. Each command has its own cluster
- Data (optional): IQ data, byte stream of interleaved IQ int16 samples.

For each received message response is sent once command is executed:

- Message: string, returns message
- Status: string, ok/error/timeout
- Response (optional): JSON formatted string, contains response parameters and data type of data field
- Data (optional): byte stream, can be interleaved IQ int16 data or json string.

This module generates events:
When messages are received, they are parsed and forwarded to corresponding module (either **Command** event or **NewData** event are generated).

For **Command** events module waits for response from HW Module before packing it into response message.

Communication module packs received IQ data into **mmWave data objects** and extracts IQ data from **mmWave data** objects before sending them as response data.

4.1.4 Data Management module

Data Management module handles **mmWave** objects.
Module manages several groups of objects independently (TX and RX group).

Module is registered for following events:

- **NewData**

- **CloseApp**
- **DM module internal event**

Objects are kept in “map” under their name.
TX bursts object names are given when **Write TX** command is executed.
RX record object names are generated based on fetch time and burst sequence which was generated.

Module tracks object creation history and deletes oldest objects in group if total memory occupied by group exceeds threshold.

TX memory threshold: 2GB
RX memory threshold: 500MB

Module provides following actions through its API (internal event):

- Get burst list – returns names of all objects in group
- Get burst – return object from group specified by name
- Clear – deletes all objects from memory

4.1.5 Log module

Log module catches events occurring in application and logs them in .log file following log4j format.

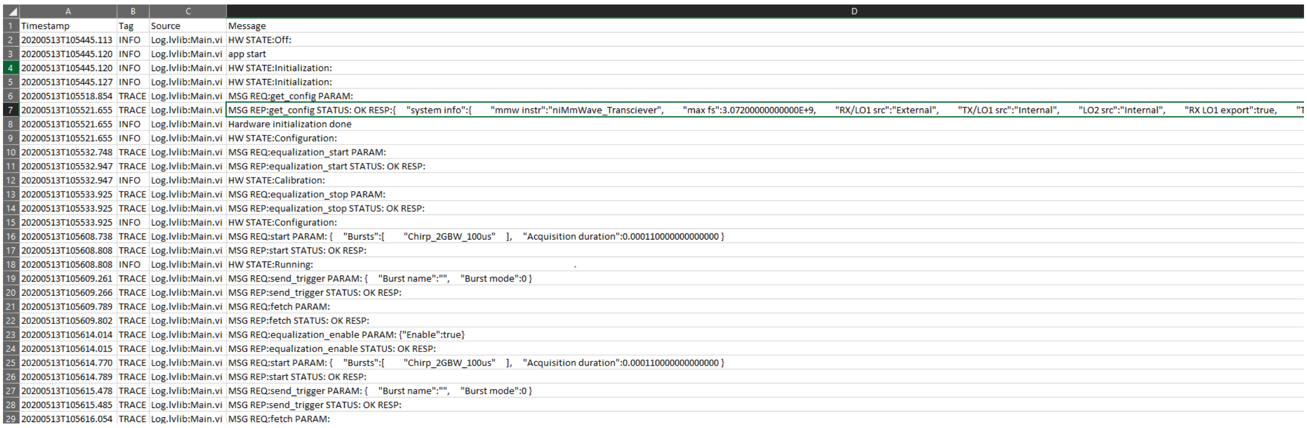
Module is registered for following events:

- **Command**
- **CloseApp**
- **Error**
- **Log module internal event**

Each log entry has timestamp.
When Command is registered log module writes command message, parameters and it also tracks response on its notifier and logs response status.

Log files are kept under: **C:/ProgramData/Noffz/mmW_instr/log**

Example log file content:



	A	B	C	D
1	Timestamp	Tag	Source	Message
2	20200513T105445.113	INFO	Log.lvlib:Main.vi	HW STATE:Off:
3	20200513T105445.120	INFO	Log.lvlib:Main.vi	app start
4	20200513T105445.120	INFO	Log.lvlib:Main.vi	HW STATE:Initialization:
5	20200513T105445.127	INFO	Log.lvlib:Main.vi	HW STATE:Initialization:
6	20200513T105518.854	TRACE	Log.lvlib:Main.vi	MSG REQ:get_config PARAM:
7	20200513T105521.655	TRACE	Log.lvlib:Main.vi	MSG REP:get_config STATUS: OK RESP:{ "system info":{ "mmw instr":"niMmWave_Transciever", "max fs":3.07200000000000E+9, "RX/LO1 src":"External", "TX/LO1 src":"Internal", "LO2 src":"Internal", "RX LO1 export":true, "T
8	20200513T105521.655	INFO	Log.lvlib:Main.vi	Hardware initialization done
9	20200513T105521.655	INFO	Log.lvlib:Main.vi	HW STATE:Configuration:
10	20200513T105532.748	TRACE	Log.lvlib:Main.vi	MSG REQ:equalization_start PARAM:
11	20200513T105532.947	TRACE	Log.lvlib:Main.vi	MSG REP:equalization_start STATUS: OK RESP:
12	20200513T105532.947	INFO	Log.lvlib:Main.vi	HW STATE:Calibration:
13	20200513T105533.925	TRACE	Log.lvlib:Main.vi	MSG REQ:equalization_stop PARAM:
14	20200513T105533.925	TRACE	Log.lvlib:Main.vi	MSG REP:equalization_stop STATUS: OK RESP:
15	20200513T105533.925	INFO	Log.lvlib:Main.vi	HW STATE:Configuration:
16	20200513T105608.738	TRACE	Log.lvlib:Main.vi	MSG REQ:start PARAM: { "Bursts":["Chirp_2GBW_100us"], "Acquisition duration":0.000110000000000000 }
17	20200513T105608.808	TRACE	Log.lvlib:Main.vi	MSG REP:start STATUS: OK RESP:
18	20200513T105608.808	INFO	Log.lvlib:Main.vi	HW STATE:Running:
19	20200513T105609.261	TRACE	Log.lvlib:Main.vi	MSG REQ:send_trigger PARAM: { "Burst name":"", "Burst mode":0 }
20	20200513T105609.266	TRACE	Log.lvlib:Main.vi	MSG REP:send_trigger STATUS: OK RESP:
21	20200513T105609.789	TRACE	Log.lvlib:Main.vi	MSG REQ:fetch PARAM:
22	20200513T105609.802	TRACE	Log.lvlib:Main.vi	MSG REP:fetch STATUS: OK RESP:
23	20200513T105614.014	TRACE	Log.lvlib:Main.vi	MSG REQ:equalization_enable PARAM: {"Enable":true}
24	20200513T105614.015	TRACE	Log.lvlib:Main.vi	MSG REP:equalization_enable STATUS: OK RESP:
25	20200513T105614.770	TRACE	Log.lvlib:Main.vi	MSG REQ:start PARAM: { "Bursts":["Chirp_2GBW_100us"], "Acquisition duration":0.000110000000000000 }
26	20200513T105614.789	TRACE	Log.lvlib:Main.vi	MSG REP:start STATUS: OK RESP:
27	20200513T105615.478	TRACE	Log.lvlib:Main.vi	MSG REQ:send_trigger PARAM: { "Burst name":"", "Burst mode":0 }
28	20200513T105615.485	TRACE	Log.lvlib:Main.vi	MSG REP:send_trigger STATUS: OK RESP:
29	20200513T105616.054	TRACE	Log.lvlib:Main.vi	MSG REQ:fetch PARAM:

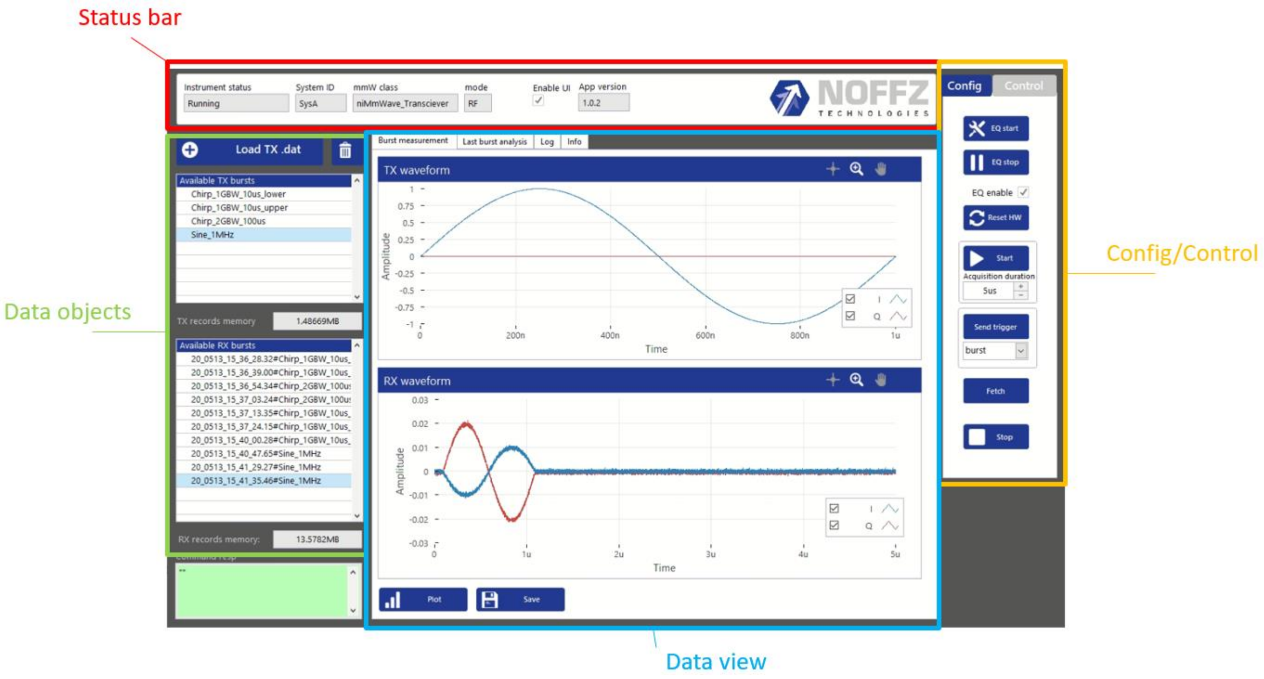
Picture 4-12 Example log file content

4.1.6 GUI Module

GUI module provides local interface to control and debug application.

Module is registered for following events:

- **CloseApp**
- **Error**



Instrument status: Running
System ID: SysA
mmW class: niMmWave_Transciever
mode: RF
Enable UI: ✓
App version: 1.0.2

Available TX bursts:
Chirp_1GBW_10us_lower
Chirp_1GBW_10us_upper
Chirp_2GBW_100us
Sine_1MHz

TX records memory: 1.48669MB

Available RX bursts:
20_0513_15_36_28.32#Chirp_1GBW_10us_
20_0513_15_36_39.00#Chirp_1GBW_10us_
20_0513_15_36_54.34#Chirp_2GBW_100us
20_0513_15_37_03.24#Chirp_2GBW_100us
20_0513_15_37_13.35#Chirp_1GBW_10us_
20_0513_15_37_24.15#Chirp_1GBW_10us_
20_0513_15_40_00.28#Chirp_1GBW_10us_
20_0513_15_40_47.65#Sine_1MHz
20_0513_15_41_29.27#Sine_1MHz
20_0513_15_41_35.46#Sine_1MHz

RX records memory: 13.5782MB

EQ enable ✓
Acquisition duration: 5us
Send trigger: burst

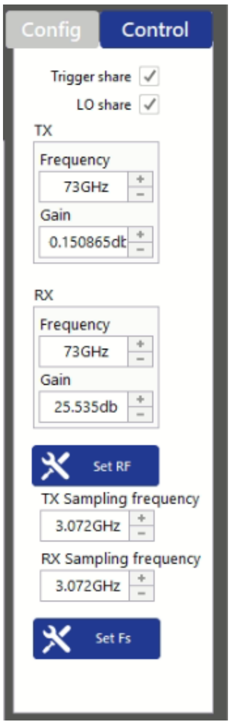
Picture 4-13 Application GUI

GUI has following main sections:

- **Status bar:** contains information regarding current state of system: HW state, mode, System ID, HAL class, version
- **Config/Control –** Controls for issuing commands to system. Has page for configuration commands and for operation commands
- **Data objects view –** Shows all TX and RX data objects currently in application

4.1.6.1 Operating instrument from GUI

When system is in configuration state from Config panel parameters can be set.



Trigger share ☑

LO share ☑

TX

Frequency 73GHz

Gain 0.150865dt

RX

Frequency 73GHz

Gain 25.535db

TX Sampling frequency 3.072GHz

RX Sampling frequency 3.072GHz

Picture 4-14 Configuration GUI

Transceiver operation:

- Select burst from **Available TX bursts** list, it will be previewed on **TX waveform** graph
- Enter desired rx record duration in **Acquisition duration** control and press **Start** button to execute **start** command and send system into **Running** state
- Selected desired burst mode from dropdown menu below **Send trigger** button (burst/continuous) and press **Send trigger** button to trigger generation and acquisition
- Press **Fetch** button to execute **fetch** command. As soon as data is retrieved new record will be added to **Available RX bursts** list
- By selecting record from **Available RX bursts** list and pressing **Plot** button selected IQ data will be plotted.

Receiver operation:

- Enter desired rx record duration in **Acquisition duration** control and press **Start** button to execute **start** command and send system into **Running** state
- If system is operating independently (no trigger sharing): Press **Send trigger** button to trigger acquisition
- If systems are time synced there is no need to press **Send trigger**. Acquisition will be triggered from other system,
- Press **Fetch** button to execute **fetch** command. As soon as data is retrieved new record will be added to **Available RX bursts** list
- By selecting record from **Available RX bursts** list and pressing **Plot** button selected IQ data will be plotted.

4.1.6.2 .dat files

When operating instrument from GUI custom IQ waveforms can be loaded into application from .dat files.

.dat files are pure binary files where I and Q samples are written in double precision format interleaved (complex double).

.dat file can be generated from complex numpy arrays or from LabVIEW (Write to binary function)

Python .dat generation example:

Create complex chirp signal and save it to .dat file

```python
import numpy as np

#Parameters
fs = 3.072e9          #sampling frequency
T = 100e-6            #signal duration
N=T*fs                #number of samples
f0=-1000e6            #staring chirp frequency
f1=1000e6             #end chirp frequency
alfa=(f1-f0)/(2*T)    #slope

#Create chirp signal
t = np.linspace(0, T, N, endpoint=False) #time vector
f = f0+alfa*t                            #frequency vector

#Save to signal to .dat
w=np.exp(1j*2*np.pi*f*t)                      #complex chirp
w.astype('complex128').tofile("C:/chirp.dat") #save to dat
```

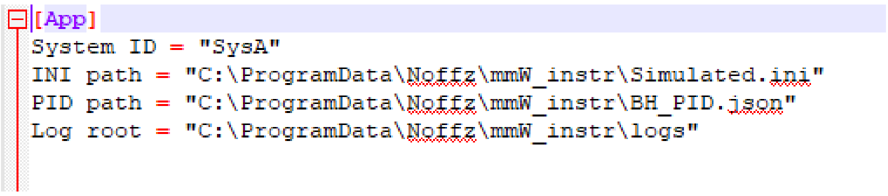
When **Load TX .dat** button is pressed popup will appear. User should navigate to folder where files are stored, and program will automatically load all .dat file from that folder.

Pressing **Save** button will save selected RX record object samples into .dat file.

4.1.7 Configuration files

Configuration files on **C:\ProgramData\Noffz\mmW_instr** are ordered into several ini files.

mmWave.ini – top level ini, specifies system ID, which hardware ini file is used, PID file path and folder where to store log files



```ini
[App]
System ID = "SysA"
INI path = "C:\ProgramData\Noffz\mmW_instr\Simulated.ini"
PID path = "C:\ProgramData\Noffz\mmW_instr\BH_PID.json"
Log root = "C:\ProgramData\Noffz\mmW_instr\logs"
```

Picture 4-15 Top level ini file content

SysA.ini – ini file for System A configuration
SysB.ini – ini file for System B configuration
Simulated.ini – ini file for Simulated operation configuration

INI files for hardware configuration have several sections and parameters. In following table details are given for complete file content

Key	Sys A value	Sys B value	Simulated value	Description
HW				

instr_class	niMmWave_Transciever_class	niMmWave_Reciever_class	Simulated	mmW HAL to be used
instr_mode	RF	RF	RF	Default operation mode
timing_class	NISync	NISync	Simulated	Timing module HAL to be used
auto_init	True	True	True	Automatically initialize app in default mode
Instruments				
TX_DAC	Tx_DAC	/	/	PXIe-3610 DAC module
TX_FPGA	Tx_FPGA	/	/	PXIe-7902R High speed serial module used for TX
IF	IF_mod_demod	IF_mod_demod	/	PXIe-3610 I/Q Generator module
RX_ADC	Rx_ADC	Rx_ADC	/	PXIe-3630 I/Q Digitizer module
RX_FPGA	Rx_FPGA	Rx_FPGA	/	PXIe-7902R High speed serial module used for RX
Timing_module	TM	TM	/	PXIe-6674T timing module
Timing module				
mode	Export trig	Import trig	/	PXI to PFI direction
Module terminal	PFI2	PFI2	/	PFI front panel terminal for trigger exporting/importing
PXI line	PXI_Trig1	PXI_Trig1	/	PXI Trigger to export/import
mmWave adapters				
TX_adapter	mmWave1/0	/	/	mmWave radio head adapter for TX (attached to IF module)
RX_adapter	mmWave0/0	mmWave0/0	/	mmWave radio head adapter for RX (attached to IF module)
RF Configuration				
TX_fc	73000000000	/	73000000000	Default TX fc
TX_gain	-10	/	-10	Default TX gain
RX_fc	73000000000	73000000000	73000000000	Default RX fc
RX_gain	25	25	25	Default RX gain
Trigger Configuration				
TX_delay	0	/	/	Delay between receiving start trigger and tx generation start
TX_trig_src	Use Tclk	/	/	Trigger source: Use TClk – internal PXI_Trig – use PXI line
TX_PXI_src	0	/	/	If PXI_Trig is source specifiec which PXI trig line to use

TX_export	None	/	/	Is start trigger exported
TX_PXI_dest	0	/	/	To which PXI line should TX trigger be exported
RX_delay	817.7087e-9	708.333e-9	/	Delay between receiving start trigger and rx acquisition start
RX_trig_src	Use Tclk	PXI_Trig	/	Trigger source: Use TClk – internal PXI_Trig – use PXI line
RX_PXI_src	0	1	/	If RX_trig_src is “PXI_Trig” which PXI trig line to use
RX_export	Synced trig	None	/	Which trigger exported: None – do not export Synced – export after Tclk sync Delayed – export after delay
RX_PXI_dest	1	0	/	To which PXI line should TX trigger be exported
Synchronization				
Sync mode	Master	Slave	/	In 2-system sync specifies which system is master (for exporting trig/LO) and which one is slave (for importing trig/LO)
Trig sync en	True	True	/	Default value for trigger synchronization enable
LO sync en	True	True	/	Default value for LO synchronization enable
LO sources				
RX/LO1 src	External	External	/	RX LO1 source
TX/LO1 src	Internal	Internal	/	TX LO1 source
LO2 src	Internal	External	/	LO2 source
LO export				
TX LO1 export	True	False	/	Export TX LO1 enable
TX LO1 export level	15	0	/	Export TX LO1 power in dBm
RX LO1 export	True	False	/	Export RX LO1enable
RX LO1 export level	15	0	/	Export RX LO1 power in dBm
LO2 export	True	False	/	Export LO2 enable
LO2 export level	13	0	/	Export LO2 power in dBm

4.1.8 Simulation mode supported features

Simulated HAL will accept all commands but some of them wont have any effect on operation (not supported)
Feature list for Simulated HAL:

Feature	Supported
Set RF	Yes
Set Fs	Yes
Write TX	Yes
Start command	Yes
Burst playlists	Yes
Arbitrary rx record duration	Yes
Send trigger	Yes
Fetch	Yes
Trigger synchronization	No
LO synchronization	No

RX record samples are generated with following logic:

- TX burst sequence to be generated is resampled from tx fs to 3.072GS/s
- Signal is then filled up with 0 samples up to rx record duration
- Signal is multiplied with complex number to apply attenuation and phase shift
- Applied attenuation is dependent on configured RF gains
- Noise signal is applied
- Signals is resampled from 3.072GS/s to rx fs.

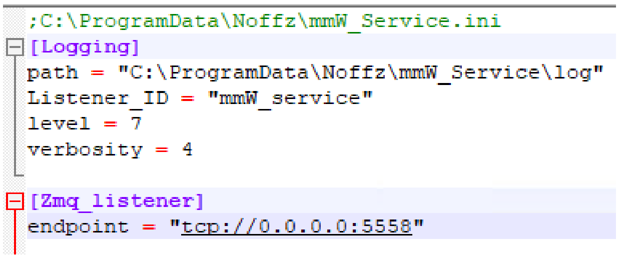
4.2 Command Service Application

Instrument control application is developed using LabVIEW 2019.

Purpose of this Service Application is to receive commands from User PC and either forward them to Instrument Application (instrument commands) or execute them (environment-control commands).
This service also provides logging: users can track and debug command in log files saved on the PXI-A system.

4.2.1 Executables, Configuration files

This service listens on the **TCP 5558** port.



```
;C:\ProgramData\Noffz\mmW_Service.ini
[Logging]
path = "C:\ProgramData\Noffz\mmW_Service\log"
Listener_ID = "mmW_service"
level = 7
verbosity = 4

[Zmq_listener]
endpoint = "tcp://0.0.0.0:5558"
```

Figure 1: Top level Config file

Accepted messages formats are described in the mmW Library documentation (Documentation - mmW Library.docx).

File locations:

- mmW_Service exe:
  - Final version (started by Windows):
    C:\ProgramData\Noffz\mmW_Service\bin\mmW_Service.exe
  - Latest build:
    <repo>\ C:\mmW_Project\mmW_Service\
- mmW_Service LabView source:
- mmW_Service shared libraries (programs used by both service and instrument):
  - mmW_commands: *<national instr.>\vi.lib\Noffz\mmW_command\mmW_commands.lvlib*
- Dynamic data (data that is genenrated dynamically and/or system specific/unique):
  - Logs: C:\ProgramData\Noffz\mmW_Service\log\*.log
  - INI file(s): C:\ProgramData\Noffz\mmW_Service\mmW_Service.INI
  - Waveforms: C:\ProgramData\Noffz\waveforms

Appendix A: Development software tools

A 1 LabVIEW tools

Following tools are needed to open and modify LabVIEW source code

Software	Type	Source
LabVIEW 2019 SP1 64- bit	Development environment	NI package manager or: https://www.ni.com/en-us/support/downloads/software-products/download.labview.html#329059
LabVIEW 2019 FPGA Module	LabVIEW module	NI package manager or: https://www.ni.com/en-us/support/downloads/software-products/download.labview-fpga-module.html#305474
FPGA Compliation Tools for LabVIEW (optional)	FPGA Compilation tools	NI package manager or: http://www.ni.com/download/labview-fpga-module-2019/8172/en/
LabVIEW modulation toolkit	LabVIEW toolkit	NI package manager or: https://www.ni.com/en-us/support/downloads/software-products/download.labview-modulation-toolkit.html#305522
NI mmWave drivers	Instrument drivers and development tools for LabVIEW	NI package manager or: https://www.ni.com/en-us/support/downloads/drivers/download.ni-mmwave.html#312718
NI Sync	Instrument drivers and development tools for LabVIEW	NI package manager or: https://www.ni.com/cs-cz/support/downloads/drivers/download.ni-sync.html
DMC GUI Suite	VI package	VI Package manager
G# Framework	VI package	VI Package manager
JKI JSON	VI package	VI Package manager
logger	VI package	VI Package manager
OpenG Toolkit	VI package	VI Package manager
ZeroMQ Socket library	VI package	VI Package manager. Package found on: http://labview-zmq.sourceforge.net/

Appendix B: Common error codes