

# **DCA1000EVM CLI Software Developer Guide**



Revision 1.01  
<10-Apr-2019>

**Revision History**

Version	Date	Author	Description
0.01	07-Mar-2019		Initial draft
0.02	25-Mar-2019		Review comments incorporated
0.03	28-Mar-2019		Review comments incorporated
1.00	29-Mar-2019		Baselined
1.01	10-Apr-2019		Reordering config option added

**Document License**

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 United States License (CC BY-SA 3.0). To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

**Contributors to this document**

Copyright (C) 2019 Texas Instruments Incorporated - <http://www.ti.com/>

# Contents

1	Introduction .....	6
1.1	Purpose & Scope .....	6
1.2	Terms / Acronyms / Abbreviations .....	6
1.3	Audience .....	7
1.4	DCA1000EVM CLI & DLL Overview .....	8
2	DCA1000EVM Architecture Overview .....	9
2.1	System Block diagram .....	9
2.2	CLI Process .....	10
2.2.1	CLI Control Tool Process .....	10
2.2.2	CLI Record Tool Process .....	12
2.2.2.1	Start Record Sequence .....	12
2.2.2.2	Stop Record Sequence .....	14
2.2.3	Inter-process Communication .....	19
2.2.3.1	Shared memory .....	20
2.2.3.2	Socket IPC .....	22
2.2.4	JSON Config File .....	23
2.2.5	Output Files .....	28
2.2.5.1	CLI logfile .....	28
2.2.5.2	Record process datafile .....	28
2.2.5.3	Record process logfile .....	29
2.2.6	System Status Codes .....	30
2.3	CLI Application Commands .....	33
2.3.1	Configure FPGA .....	34
2.3.2	Configure EEPROM .....	36
2.3.3	Reset FPGA .....	40
2.3.4	Reset RADAR EVM .....	42
2.3.5	Start record .....	45
2.3.6	Stop record .....	48
2.3.6.1	User asynchronous stop record .....	48
2.3.6.2	DCA1000EVM asynchronous status stop record .....	51

2.3.7 Configure record delay .....	53
2.3.8 Read DLL version.....	56
2.3.9 Read FPGA version .....	57
2.3.10 Query record process status .....	60
2.3.11 Query system aliveness status.....	62
2.4 DLL .....	66
2.4.1 Commands Protocol .....	66
2.4.1.1 Request packet protocol.....	66
2.4.1.2 Response packet protocol .....	66
2.4.1.3 Data packet protocol .....	66
2.4.2 Callback Functions .....	67
2.4.2.1 Asynchronous status callback.....	67
2.4.2.2 Record process status callback .....	68
2.4.3 Threads .....	69
2.4.4 Record data stop mode processing .....	70
2.4.5 Record data processing.....	71
2.4.5.1 Post processing .....	71
2.4.5.2 Inline processing .....	74
2.5 FPGA .....	77
3 Software Reference .....	82

## 1 Introduction

This document gives a detailed description of the software architecture of the DCA1000EVM CLI application. This document explains the interfaces and modules involved in the DCA1000EVM CLI application, FPGA communication and DLL level design.

### 1.1 Purpose & Scope

The purpose of this document is to present a detailed interfaces and modules description of the DCA1000EVM CLI software application and FPGA and DLL interface levels. The scope of this document is to specify the structure and design of the modules discussed. The associated hardware functionalities are not elaborated in detail.

### 1.2 Terms / Acronyms / Abbreviations

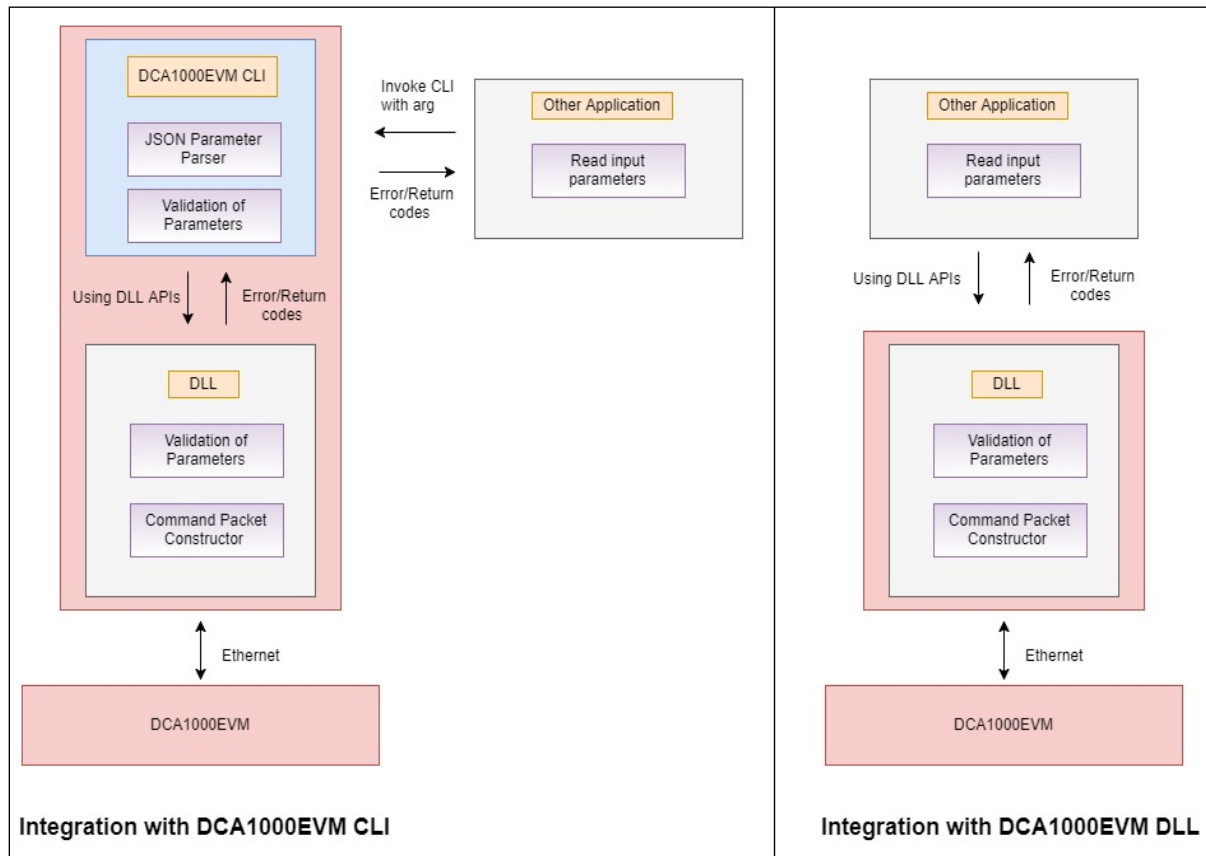
ACK	Acknowledgement
ADC	Analog-to-Digital Converter
API	Application Programming Interface
CLI	Command Line Interface
DDR	Double Data Rate
DLL	Dynamic Link Library
DOS	Disk Operating System
EEPROM	Electrically Erasable Programmable Read-Only Memory
EVM	Evaluation Module
FPGA	Field Programmable Gate Array
GB	Gigabit
GUI	Graphical User Interface
IP	Internet Protocol
IPC	Inter Process Communication
IPV4	Internet Protocol Version 4
JSON	JavaScript Object Notation
LIB	Library
LSB	Least Significant Bit
LTS	Long Term Support
LVDS	Low-Voltage Differential Signaling
MAC	Media Access Control

Mbps	Megabits per Second
MSB	Most Significant Bit
NACK	Negative Acknowledgement
PC	Personal Computer
OS	Operating System
OSAL	Operating System Abstraction Layer
UDP	User Datagram Protocol
UI	User Interface
us	Micro Second

### 1.3 Audience

Anyone interested in understanding the DCA1000EVM CLI design.

## 1.4 DCA1000EVM CLI & DLL Overview



**Figure 1 DCA1000EVM CLI & DLL Overview**

The DCA1000EVM CLI and DLL application performs configuration of DCA1000EVM and recording based on the user inputs. Control commands will block waiting for the response till it gets timed out. The user input can be Linux shell or a system call from another application or DOS prompt. CLI and DLL supports multiple instances if unique port numbers are used.

There are two modes for application to integrate the DCA1000EVM functionality.

➤ Interacting with CLI

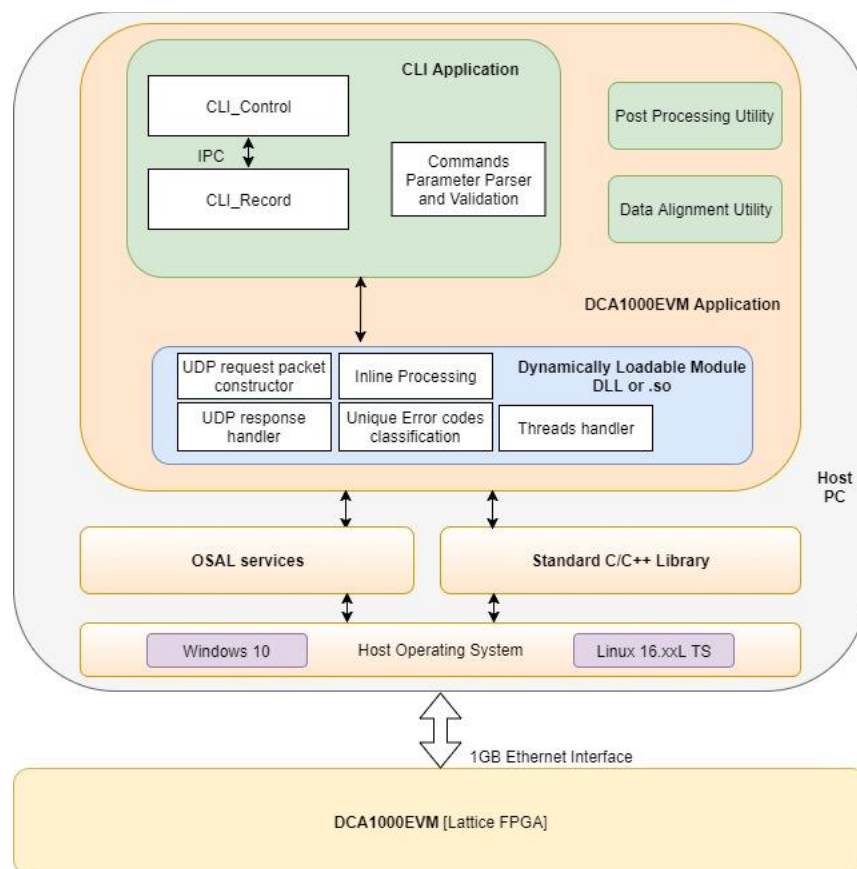
- Application shall send parameters in JSON formatted config file for the corresponding CLI Control commands and shall support multiple data capture without closing the host application.
- CLI application supports both Windows and Linux
- Acknowledgement of commands and error codes are handled and returned by the CLI application as a signed 32-bit integer value to the calling application. Calling application can also read the last error code status returned by the CLI process using *GetLastError()* in windows and *errno* status in Linux.
- The response status of each of the command is captured in a log file by the CLI application.
- CLI application supports running as a foreground as well as a background process.



- Interacting with DLL
  - Application shall send parameters in defined structure as arguments for corresponding APIs and register a callback function for handling async status packets and shall support multiple data capture without closing the host application.
  - DLL supports both Windows (as dynamic library [.dll]) and Linux (as shared library [.so])
  - Acknowledgement of commands and error codes are handled and returned by the DLL application as a signed 32-bit integer value to the calling application. Calling application can also read the last error code status occurred in the DLL calls using *GetLastError()* in windows and *errno* status in Linux.

## 2 DCA1000EVM Architecture Overview

### 2.1 System Block diagram



**Figure 2 DCA1000EVM CLI Block Diagram**

#### CLI Application:

- DCA1000EVM CLI application consists of 2 CLI tools to perform configuration of DCA1000EVM system and recording of data in files.
- CLI\_Control tool invokes the CLI Record tool on *start\_record* and then communicates with CLI\_Record tool using socket IPC for passing *stop\_record* message to initiate stop recording.
- Control and Record tools use the same JSON input configuration file to retrieve the setup parameters.

- OSAL services like shared memory, socket and timer event are handled as a separate module for Windows and Linux platforms.

**DLL Module:**

- DLL receives its configuration via set of APIs and uses that information to communicate with DCA1000EVM system over Ethernet using UDP protocol.
- DLL supports 2 modes of record processing. Both record processing modes are mutually exclusive, and DLL need to be compiled with any one of the processing modes. DLL does not support runtime changing of record processing mode.
  - Post processing – Data will be captured and stored in the file directly. Offline, the captured data can be processed for zero filling of dropped packets and sorting of packets using post processing utility. Then the processed data can be realigned using data alignment utility.
  - Inline processing – Data will be captured and stored in the buffer. In parallel, the buffered data will be processed for zero filling of dropped packets and sorting of packets. Then the processed data will be realigned and stored in the files.

**2.2 CLI Process**

CLI application suite includes two executables.

- DCA1000EVM\_CLI\_Control.exe
- DCA1000EVM\_CLI\_Record.exe

**2.2.1 CLI Control Tool Process**

The DCA1000EVM\_CLI\_Control.exe is the application providing options for the configuration commands and initiating the start and stop of the record process (DCA1000EVM\_CLI\_Record.exe). Following are some of the key aspects of the DCA1000EVM\_CLI\_Control tool.

- CLI\_Control tool supports blocking calls for configuration and Stop Record commands.
- CLI\_Control tool supports non-blocking for the Start Record and record *query\_status* commands.
- Multiple instances of the CLI\_Control tool can be instantiated on the PC to control multiple EVMs while using unique UDP Ports for each EVM.
- It provides option to execute in “quiet” mode wherein there are no messages being displayed on the console.
- For all configuration commands, the configJsonFile should have valid DCA1000EVM system IP and config/record port numbers and the parameters corresponding to the executing commands.
- When CLI\_Control is invoked to send continuous commands, CLI\_Control will execute the command only when no other commands are in execution. If any other command is in progress, the CLI\_Control will prompt the user to stop the already running process.

The generic execution flow of CLI\_Control commands is as follows,

- User will initiate the control command through script or command line as  
*DCA1000EVM\_CLI\_Control <command> configFile.json*

- CLI\_Control tool reads the shared memory for checking whether any record process is running.
- If record process is running, CLI\_Control tool will prompt the user to stop the already running record process. The CLI\_Control tool logs the information in a log file with timestamp and will be terminated.
- If no process is running, CLI\_Control tool validates the input parameters from the configJsonfile.
- CLI\_Control initializes the config port ethernet connection and send <command> and wait for response packet till timeout.
- CLI\_Control displays the command status to the user in the console. The CLI\_Control tool also stores the configuration command status in a log file with timestamp. The exit code of the last executed process can also be read using OS system calls based on the calling application.

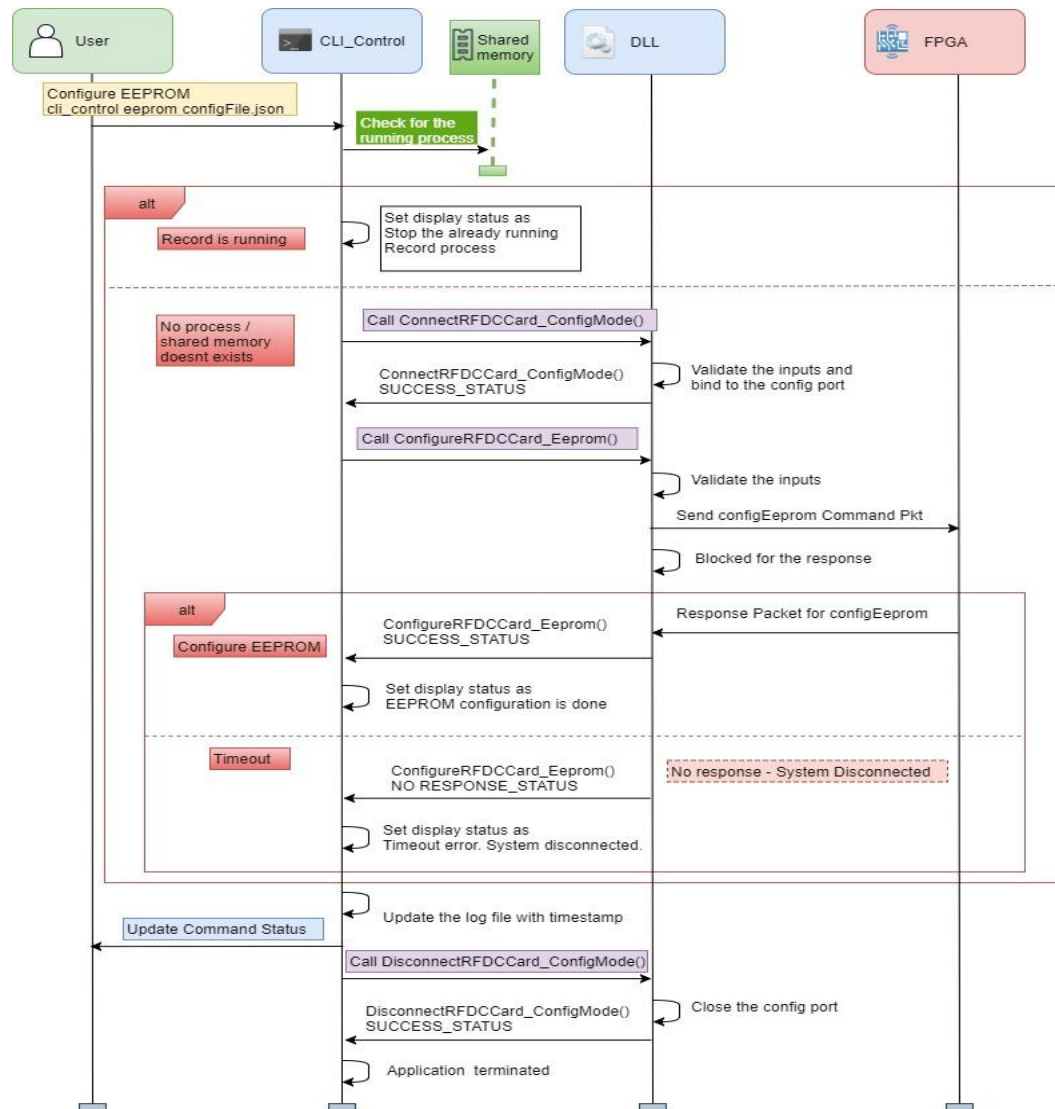


Figure 3 CLI Control Tool Process

## 2.2.2 CLI Record Tool Process

DCA1000EVM CLI Record tool handles record process by interacting with the DCA1000EVM over Ethernet connection.

The generic execution flow of CLI\_Record tool process is as follows,

- CLI\_Control tool initiates the Start record process by invoking CLI\_Record tool. CLI\_Control will poll the shared memory for start command status and display the command status in the console and stores it in log file with the timestamp. Then it terminates.
- CLI\_Record tool initializes the config and data ports ethernet connection.
- CLI\_Record tool handles the responses for start and stop commands and async packets from DCA1000EVM and update the shared memory and in the console/logfile based on the running mode of the record process.
- CLI\_Record tool start capturing data on successful start command response.
- For inline and post processing, CLI\_Record tool updates the status of the dropped packets metadata in the shared memory whenever any out of sequence packet is received through Ethernet.
- When recording is in progress, other control commands cannot be executed using CLI\_Control tool until stop command is issued. CLI\_Control tool will prompt the user as 'Stop the already running record process' by reading status in shared memory.
- CLI\_Control tool initiates the Stop record process by messaging the CLI\_Record tool through socket IPC. CLI\_Control will send stop command to the config port which is listened by CLI\_Record tool. Then CLI\_Record will send stop command to DCA1000EVM and updates the shared memory and in the console/logfile based on the running mode of the record process.
- CLI\_Control will poll the shared memory for stop command status and display the command status in the console and stores it in log file along with timestamp.
- CLI\_Record tool will overwrite at any point of the record process whenever data is transferred to or from DCA1000EVM like async and start/stop config commands.
- CLI\_Control tool can poll shared memory for record process at any point of time. CLI\_Control will only read the shared memory. CLI\_Record tool will only write the shared memory.

Start and stop record process are detailed in the below section.

### 2.2.2.1 Start Record Sequence

- CLI\_Control tool will verify the record process status in shared memory.

- If any record process is in running state, CLI\_Control tool will inform the user to stop the already running record process and will be terminated.
- If no record process is running, CLI\_Control tool will proceed through following steps.
- CLI\_Control tool will verify the DCA1000EVM connectivity.
  - If the system is not connected, CLI\_Control tool will inform the user that the system is disconnected and will be terminated.
  - If the system is connected, CLI\_Control tool will proceed through following steps.
- CLI\_Control tool will create the shared memory based on the unique config port and invoke the CLI\_Record tool.
- CLI\_Record tool will bind to the config port and data ports.
- CLI\_Record tool will send the start command to DCA1000EVM and handle its response.
  - On successful start command response, record data will be captured in the files and CLI\_Record tool will update the status in shared memory as **"Record is in progress"**
  - On failure/no response of start command, CLI\_Record tool will update the status in shared memory as **"Start Command Failure/No Response"**. CLI\_Record tool will be terminated.
- CLI\_Control tool will poll for the start command status for a defined time from shared memory.
- CLI\_Control tool will update the start command status in the console and in the logfile and will be terminated.

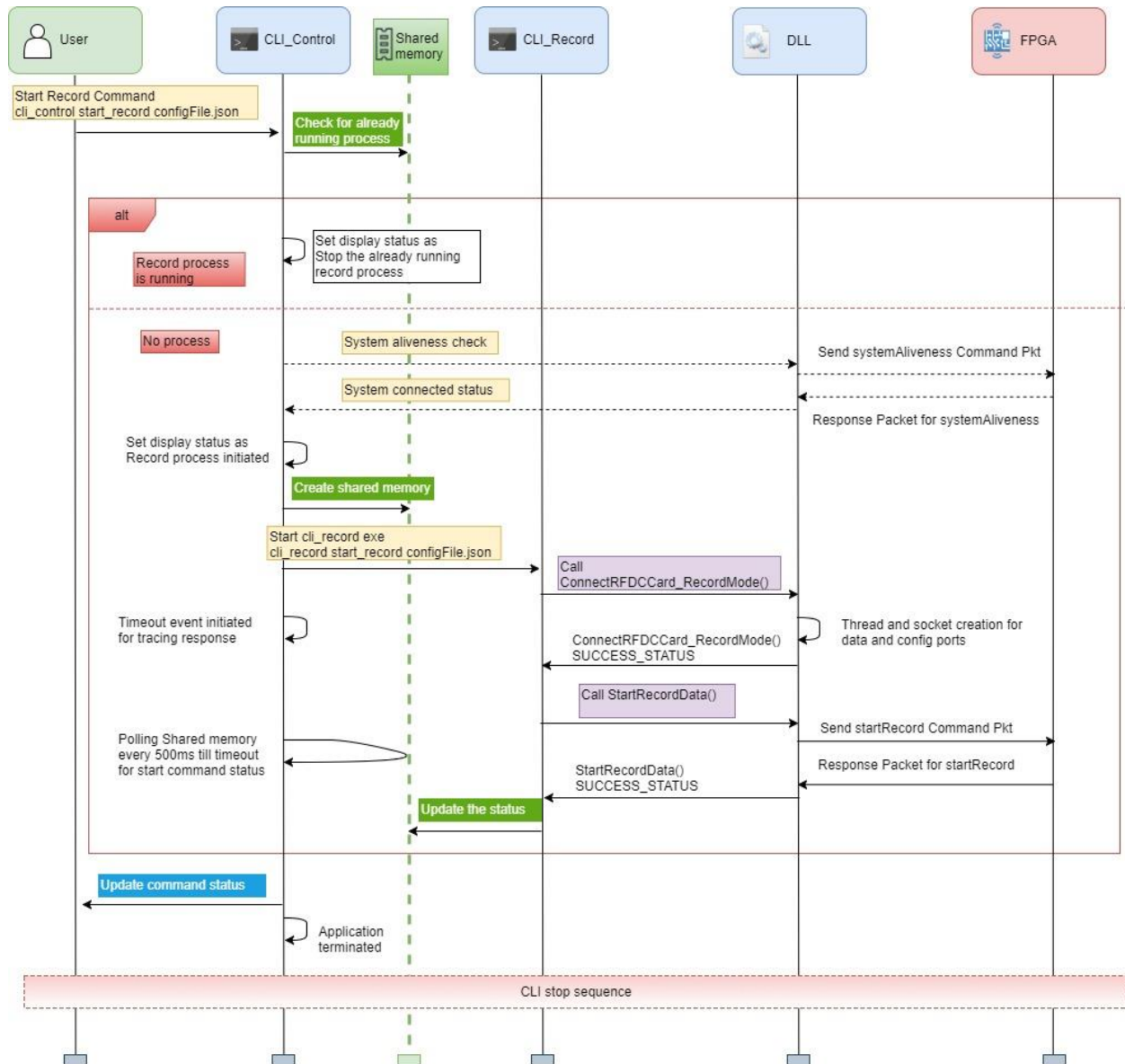


Figure 4 CLI Record Tool Process – Start Sequence

## 2.2.2.2 Stop Record Sequence

### 2.2.2.2.1 User asynchronous stop record

- CLI\_Control tool will verify the record process status in shared memory.
  - If no record process is running, CLI\_Control tool will inform the user that no process is running to stop and will be terminated.
  - If any record process is in running state, CLI\_Control tool will proceed through following steps.

- CLI\_Control tool will use DLL to pass the stop message to CLI\_Record tool over the local config port to stop recording.
- On receiving the message, CLI\_Record tool will use the DLL to send the stop command to DCA1000EVM over the config port to stop the record data streamed over Ethernet from the DCA1000EVM.
  - On successful stop command response, CLI\_Record tool will read all the data in the socket buffer and update the status in shared memory as “**Record Stopped**”. CLI\_Record tool will be terminated.
  - On failure/no response of stop command, CLI\_Record tool will update the status in shared memory as “**Stop Command Failure/No Response**”. CLI\_Record tool will be terminated.
- CLI\_Control tool will poll for the stop command status for a defined time from shared memory.
- CLI\_Control tool will update the stop command status in the console and in the logfile and will be terminated.

## DCA1000EVM Architecture Overview

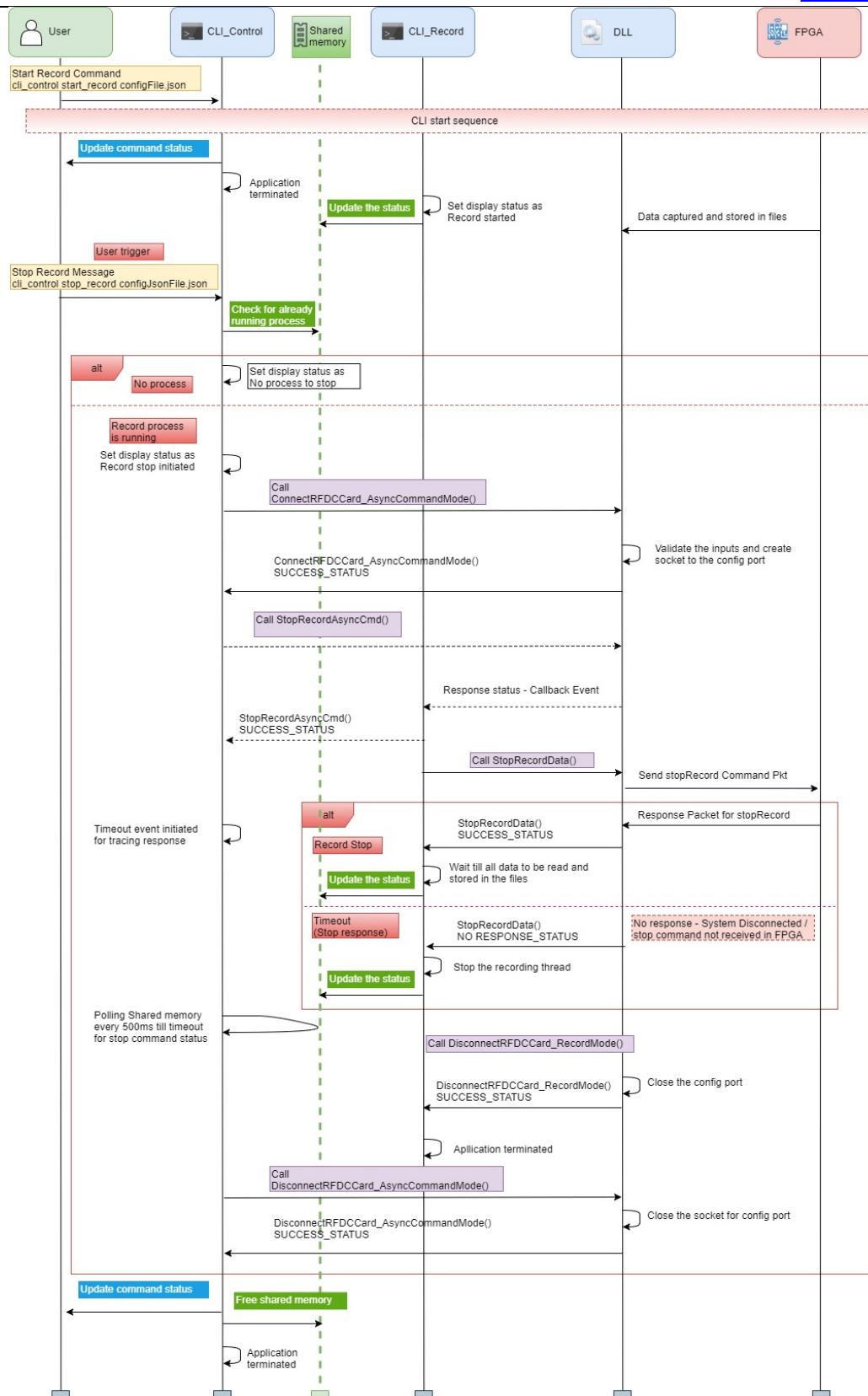
[www.ti.com](http://www.ti.com)


Figure 5 CLI Record Tool – User Asynchronous Stop Record Sequence



#### 2.2.2.2.2 DCA1000EVM asynchronous status stop record

- The following asynchronous status from DCA1000EVM initiates the stop record command
  - No LVDS data status
  - No header status
  - Record completed status
- On receiving any one of the above asynchronous status from DCA1000EVM, CLI\_Record tool will use the DLL to send the stop command to DCA1000EVM over the config port to stop the record data streamed over Ethernet from the DCA1000EVM.
  - On successful stop command response, CLI\_Record tool will read all the data in the socket buffer and update the status in shared memory as **“Record Stopped”**. CLI\_Record tool will be terminated.
  - On failure/no response of stop command, CLI\_Record tool will update the status in shared memory as **“Stop Command Failure/No Response”**. CLI\_Record tool will be terminated.

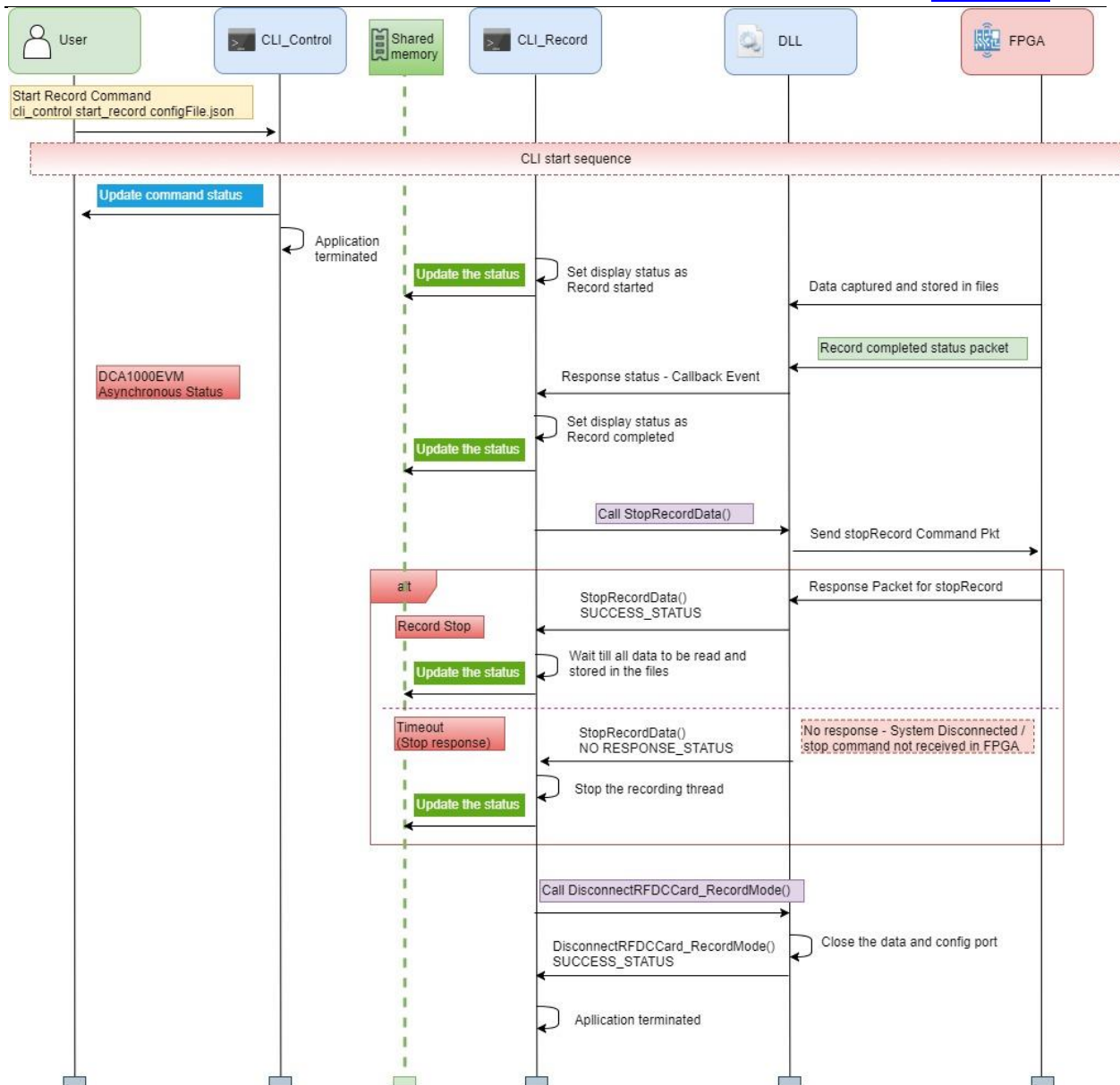


Figure 6 CLI Record Tool – DCA1000EVM Asynchronous Status Stop Record

### 2.2.2.2.3 DLL asynchronous status stop record

- The following asynchronous status from DLL initiates the stop record command
  - Timeout error status
  - Record data file creation error status
  - Reordering error status (inline processing)

- On receiving any one of the above asynchronous status from DLL, CLI\_Record tool will use the DLL to send the stop command to DCA1000EVM over the config port to stop the record data streamed over Ethernet from the DCA1000EVM.
  - On successful stop command response, CLI\_Record tool will read all the data in the socket buffer and update the status in shared memory as **“Record Stopped”**. CLI\_Record tool will be terminated.
  - On failure/no response of stop command, CLI\_Record tool will update the status in shared memory as **“Stop Command Failure/No Response”**. CLI\_Record tool will be terminated.

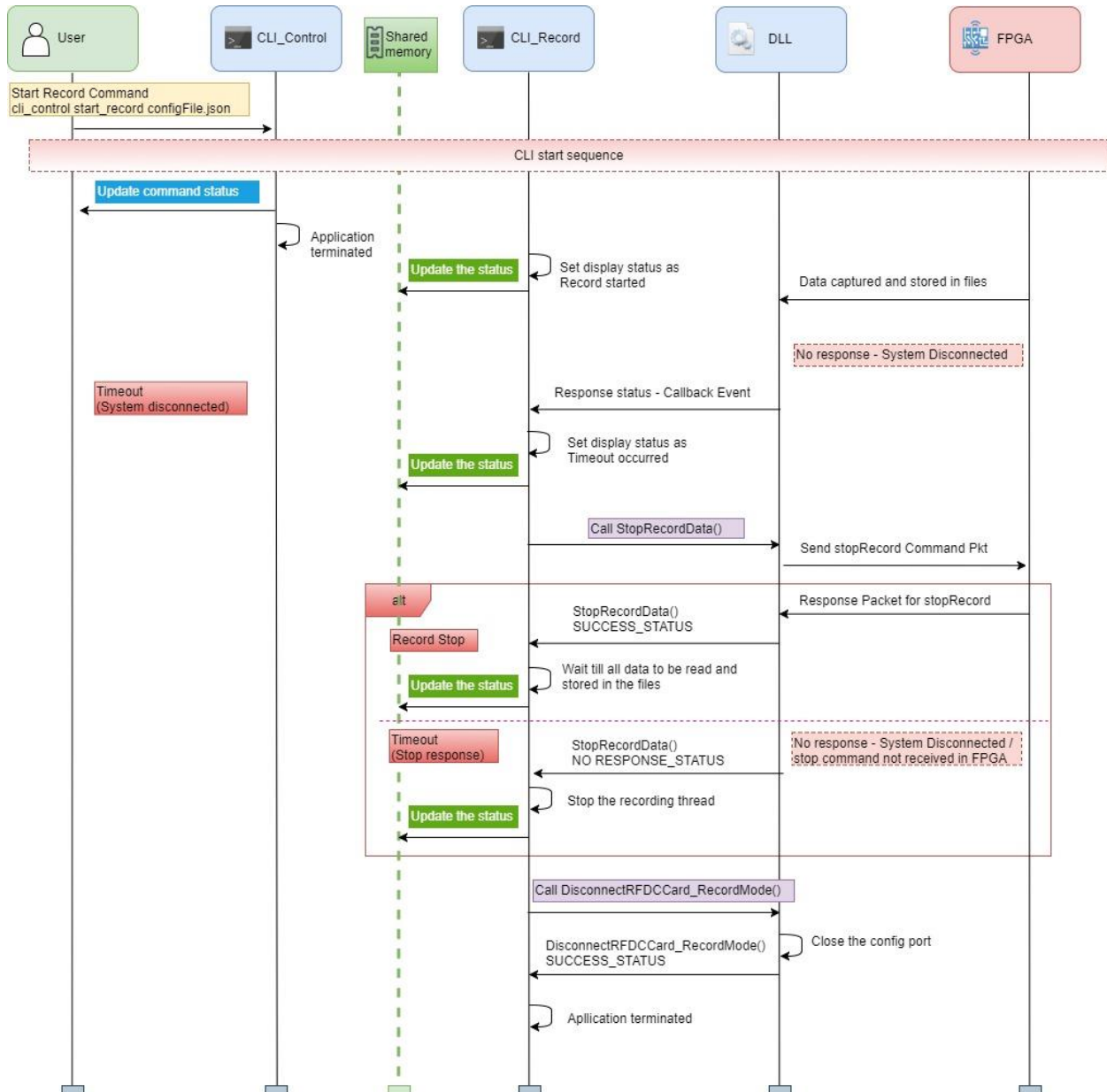


Figure 7 CLI Record Tool – DLL Asynchronous Status Stop Record

### 2.2.3 Inter-process Communication

Shared memory is used to exchange information such as command status and states between CLI\_Record and CLI\_Control tools. Socket IPC is used for execution of control commands with DCA1000EVM system and for sending the asynchronous stop record command from CLI\_Control tool to CLI\_Record tool.

### 2.2.3.1 Shared memory

- Shared memory is used as IPC mechanism between CLI\_Control and CLI\_Record tool to read the command status and states of CLI application.
- CLI\_Control tool will create the shared memory during successful start of CLI\_Record tool.
- CLI\_Control tool will free the shared memory during the stop of CLI\_Record process.
- CLI\_Record tool will write their different states in the shared memory which will be read by CLI\_Control tool at any point of time.

Record Process Status (s32CommandStatus)	Status	Description
STS_CLI_REC_PROC_IS_IN_PROG	-4029	When the record start command response is received by CLI_Record tool, CLI_Record tool will update the shared memory state as 'Record is in progress'
STS_CLI_REC_PROC_STOPPED	-4030	When the record stop command response is received by CLI_Record tool, CLI_Record tool will update the shared memory state as 'Record process is stopped'
STS_CLI_REC_PROC_START_INIT	-4031	When the record start command is executed by CLI_Record tool, CLI_Record tool will update the shared memory state as 'Record process is initiated'
STS_CLI_REC_PROC_START_FAILED	-4032	When the record start command response is not received by CLI_Record tool, CLI_Record tool will update the shared memory state as 'Start record process is failed'
STS_CLI_REC_PROC_STOP_INIT	-4033	When the record stop command is executed by CLI_Record tool, CLI_Record tool will update the shared memory state as 'Stop record process is initiated'
STS_CLI_REC_PROC_STOP_FAILED	-4069	When the record stop command response is not received by CLI_Record tool, CLI_Record tool will update the shared memory state as 'Stop record process is failed'

**Table 1 Record process states**

- Shared memory address name is assigned based on config port.
  - For e.g.,
    - clishm\_<config\_port> i.e clishm\_4096
- Shared memory shall have the following structure [if Record is in progress]

```
typedef struct
{
    /** Command Code of the record command */
    UINT16 u16CommandCode;

    /** Command status of the record process */
    SINT32 s32CommandStatus;

    /** Async DCA1000EVM status [if Record is in progress]
     * Each bit can be set for the defined async status from DCA1000EVM
     * Refer section 2.2.6 for system async status updated in the shared memory
     */
    UINT32 u32AsyncStatus;

    /** Inline process summary */
    strRFDCCard_InlineProcStats strInlineProcStats;

}SHM_PROC_STATES;

/** Inline processing statistics */
typedef struct
{
    /** Data type header ID */
    SINT8 s8HeaderId[NUM_DATA_TYPES][MAX_NAME_LEN];

    /** First packet ID */
    UINT32 u32FirstPktId[NUM_DATA_TYPES];

    /** Last packet ID */
    UINT32 u32LastPktId[NUM_DATA_TYPES];

    /** Out of sequence count */
    ULONG64 u64OutOfSeqCount[NUM_DATA_TYPES];

    /** Received packets count */
    ULONG64 u64NumOfRecvdPackets[NUM_DATA_TYPES];

    /** Zero filled packets count */
    ULONG64 u64NumOfZeroFilledPackets[NUM_DATA_TYPES];

    /** Zero filled bytes count */
    ULONG64 u64NumOfZeroFilledBytes[NUM_DATA_TYPES];

    /** Packet start timestamp */
    time_t StartTime[NUM_DATA_TYPES];
}
```

```

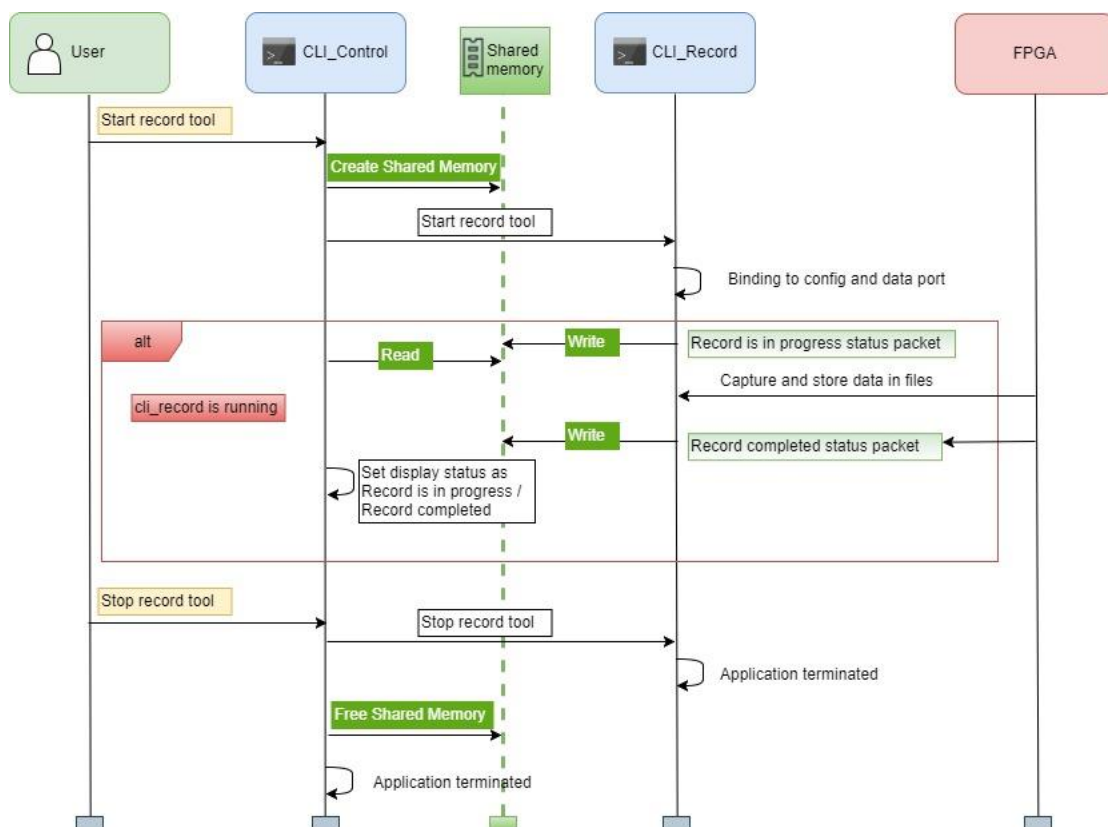
/** Packet end timestamp */
time_t EndTime[NUM_DATA_TYPES];

/** Packet out of sequence seen from offset */
UINT32 u32OutOfSeqPktFromOffset[NUM_DATA_TYPES];

/** Packet out of sequence seen till offset */
UINT32 u32OutOfSeqPktToOffset[NUM_DATA_TYPES];

}strRFDCCard_InlineProcStats;

```



**Figure 8 Shared Memory - Write process states**

Refer section [2.3.10](#) for more details on how CLI Control tool will read the shared memory.

### 2.2.3.2 Socket IPC

- Socket is used as IPC mechanism by CLI\_Control tool to pass the stop message from the user to the existing CLI\_Record tool.
- CLI\_Control will send the following command structure to the local config port of the running PC. CLI\_Record tool will listen to the command and send the stop command to DCA1000EVM and handle its response.

- After successful/no response from DCA1000EVM, CLI\_Record tool will update the status in shared memory which will be polled by CLI\_Control tool till the command gets timeout/get any response.

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x24	-	-	Command code for async record stop from CLI
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

**Table 2 FPGA command structure – Stop Record Async Command**

## 2.2.4 JSON Config File

CLI tool accepts configuration parameter through JSON format config file. The config file is given as a command line argument to the CLI tool.

File extension : .json

Type : JSON formatted data in text format

Sample JSON File Data:

```
{
  "DCA1000Config": {
    "dataLoggingMode": "raw",
    "dataTransferMode": "LVDS capture",
    "dataCaptureMode": "ethernetStream",
    "lvdsMode": 1,
    "dataFormatMode": 1,
    "packetDelay_us": 25,
    "ethernetConfig": {
      "DCA1000IPAddress": "192.168.33.180",
      "DCA1000ConfigPort": 4096,
      "DCA1000DataPort": 4098
    },
    "ethernetConfigUpdate": {
      "systemIPAddress": "192.168.33.30",
      "DCA1000IPAddress": "192.168.33.180",
      "DCA1000MACAddress": "12.34.56.78.90.12",
      "DCA1000ConfigPort": 4096,
      "DCA1000DataPort": 4098
    }
  },
}
```

```

"captureConfig": {
    "fileBasePath": "D:\\capture",
    "filePrefix": "outdoor_capture",
    "maxRecFileSize_MB": 1024,
    "sequenceNumberEnable": 1,
    "captureStopMode": "duration",
    "bytesToCapture": 50000,
    "durationToCapture_ms": 5000,
    "framesToCapture": 10
},
"dataFormatConfig": {
    "MSBToggle": 0,
    "reorderEnable": 1,
    "laneFmtMap": 0,
    "dataPortConfig": [
        {
            "portIdx": 0,
            "dataType": "real"
        },
        {
            "portIdx": 1,
            "dataType": "complex"
        },
        {
            "portIdx": 2,
            "dataType": "real"
        },
        {
            "portIdx": 3,
            "dataType": "real"
        },
        {
            "portIdx": 4,
            "dataType": "complex"
        }
    ]
}
}

```

The min, max and default values for the JSON config file parameters are as follows:

Parameter	Min	Max	Default	Description
DCA1000Config	-	-	-	Object contains DCA1000EVM related configuration



dataLoggingMode	-	-	Raw	<p>Data logging mode specifies the type of data being transferred in record mode through DCA1000EVM. This field is valid only when dataTransferMode is "LVDS Capture".</p> <p>The valid options are</p> <ul style="list-style-type: none"> <li>• raw</li> <li>• multi</li> </ul>
dataTransferMode	-	-	LVDS Capture	<p>Data transfer mode specifies if DCA1000EVM is in record mode or playback mode.</p> <p>The valid options are</p> <ul style="list-style-type: none"> <li>• LVDS Capture</li> <li>• LVDS Playback</li> </ul>
dataCaptureMode	-	-	ethernetStream	<p>Data capture mode specifies the transport mechanism for getting data out of DCA1000EVM. This field is valid only when dataTransferMode is "LVDS Capture".</p> <p>The valid options are</p> <ul style="list-style-type: none"> <li>• ethernetStream</li> <li>• SDCardStorage</li> </ul>
lvdsMode	1	2	1	<p>LVDS mode specifies the lane config for LVDS. This field is valid only when dataTransferMode is "LVDS Capture".</p> <p>The valid options are</p> <ul style="list-style-type: none"> <li>• 1 (4lane)</li> <li>• 2 (2lane)</li> </ul>
dataFormatMode	1	3	3	<p>Data format mode specifies the bit-mode for the captured data. This field is valid only when dataTransferMode is "LVDS Capture".</p> <p>The valid options are</p> <ul style="list-style-type: none"> <li>• 1 (12 bit)</li> <li>• 2 (14 bit)</li> <li>• 3 (16 bit)</li> </ul>

## DCA1000EVM Architecture Overview

[www.ti.com](http://www.ti.com)

packetDelay_us	5	500	25	Value in usec to throttle the throughput of the Ethernet stream out of DCA1000EVM. Min and max values are dictated by the limits supported by DCA1000 H/W. This field is valid only when dataCaptureMode is "ethernetStream"
ethernetConfig	-	-	-	Config block for Ethernet stream. This block is valid only when dataCaptureMode is "ethernetStream"
DCA1000IPAddress	-	-	192.168.33.180	IP address of the DCA1000EVM
DCA1000ConfigPort	1	65535	4096	Config port number for config command communication between DCA1000EVM and PC
DCA1000DataPort	1	65535	4098	Data port number for data communication between DCA1000EVM and PC
ethernetConfigUpdate	-	-	-	Config block to reconfigure the Ethernet details in EEPROM of DCA1000EVM
systemIPAddress	-	-	192.168.33.30	To reconfigure the IP address of the PC in EEPROM of DCA1000EVM
DCA1000IPAddress	-	-	192.168.33.180	To reconfigure the IP address of the DCA1000EVM in EEPROM of DCA1000EVM
DCA1000MACAddress	-	-	12.34.56.78.90.12	To reconfigure the MAC address of the DCA1000EVM in EEPROM of DCA1000EVM
DCA1000ConfigPort	1	65535	4096	To reconfigure the config port number in EEPROM of DCA1000EVM for config command communication between DCA1000EVM and PC
DCA1000DataPort	1	65535	4098	To reconfigure the data port number in EEPROM of DCA1000EVM for data communication between DCA1000EVM and PC
captureConfig	-	-	-	Config block for data capture
fileBasePath	-	-	D:\capture	Valid file path on the PC where this CLI runs.

				This block is valid only when dataCaptureMode is "ethernetStream"
filePrefix	-	-	outdoor_capture	Filename conforming to host PC rules; CLI would append index numbers, etc to this filePrefix to derive the actual filename that contains recorded data
maxRecFileSize_MB	1	1024	1024	Record data file maximum size in MB
sequenceNumberEnable	0	1	0	<p>This field controls whether the packet sequence number need to be stored in the data file or not. This field is valid only when data captured using post processing method.</p> <p>The valid options are</p> <ul style="list-style-type: none"> <li>• 0 (Disable)</li> <li>• 1 (Enable)</li> </ul>
captureStopMode	-	-	infinite	<p>Stop mode for the capture. Based on this config, other fields should be specified to provide more config for that capture stop mode.</p> <p>The valid options are</p> <ul style="list-style-type: none"> <li>• bytes</li> <li>• frames</li> <li>• duration</li> <li>• infinite</li> </ul>
bytesToCapture	128	0xFFFF FFFF	50000	Specifies the number of bytes to capture when captureStopMode is "bytes"
durationToCapture_ms	40	0xFFFF FFFF	5000	Specifies the capture duration in msec when captureStopMode is "duration"
framesToCapture	1	0xFFFF	10	Specifies the number of radar frames to capture when captureStopMode is "frames"
dataFormatConfig	-	-	-	Config block specifies the data format to assist in data formatting services of the CLI
MSBToggle	0	1	0	Specifies the MSB toggle in the captured data (16-bit value) to be enabled or not

				The valid options are <ul style="list-style-type: none"> <li>• 0 (Disable)</li> <li>• 1 (Enable)</li> </ul>
reorderEnable	0	1	1	Specifies the reordering of the captured data (bytes) to be enabled or not  The valid options are <ul style="list-style-type: none"> <li>• 0 (Disable)</li> <li>• 1 (Enable)</li> </ul>

**Table 3 JSON File data description**

## 2.2.5 Output Files

### 2.2.5.1 CLI logfile

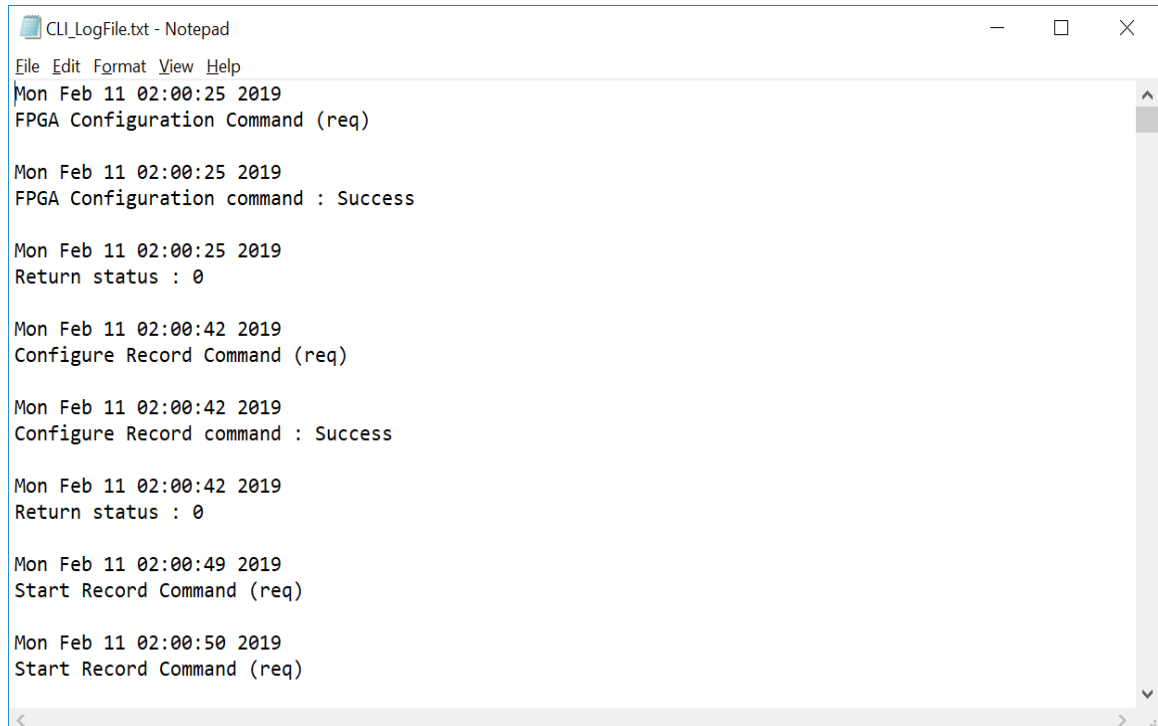
File name : *CLI\_LogFile.txt*

File extension : .txt

Type : Text

Description :

Contains all the commands execution information along with the timestamp. It can be viewed manually whenever required. The fill will be appended for new sessions and new command execution information.



```

CLI_LogFile.txt - Notepad
File Edit Format View Help
Mon Feb 11 02:00:25 2019
FPGA Configuration Command (req)

Mon Feb 11 02:00:25 2019
FPGA Configuration command : Success

Mon Feb 11 02:00:25 2019
Return status : 0

Mon Feb 11 02:00:42 2019
Configure Record Command (req)

Mon Feb 11 02:00:42 2019
Configure Record command : Success

Mon Feb 11 02:00:42 2019
Return status : 0

Mon Feb 11 02:00:49 2019
Start Record Command (req)

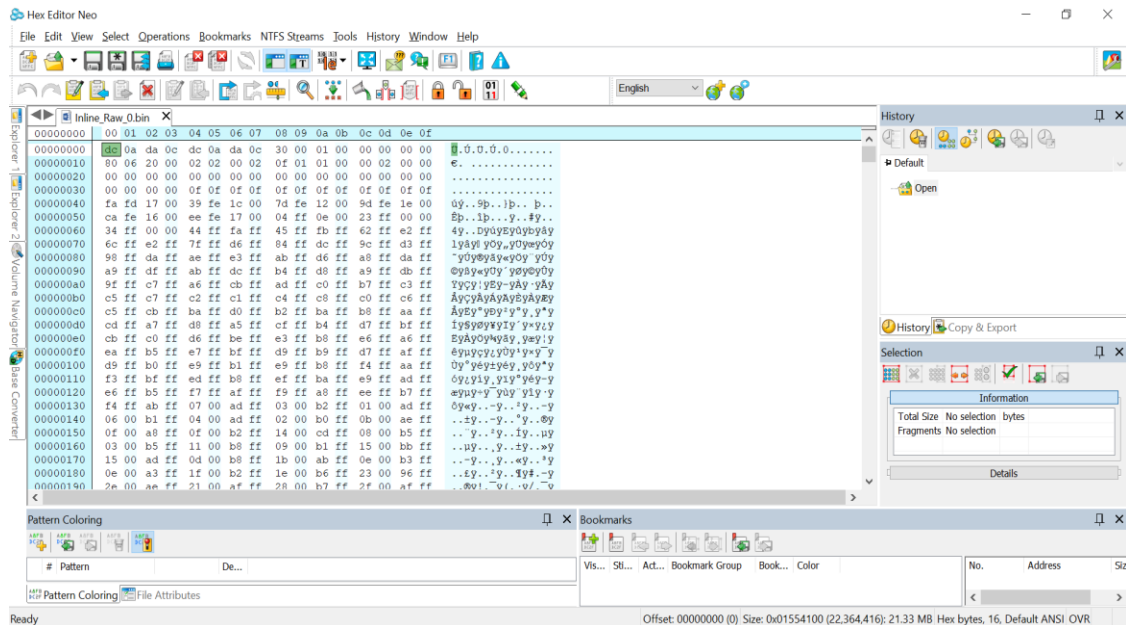
Mon Feb 11 02:00:50 2019
Start Record Command (req)
  
```

### 2.2.5.2 Record process datafile

- “RawModeCap\_Raw\_0.bin” → Raw data would be captured

- "MultiModeCap\_hdr\_xxxx\_0.bin" -> xxxx refers to 4 characters of the header received in the first packet of the respective 5 ports.

For ADC data, filename will be generated as “MultiModeCap\_hdr\_0ADC\_0.bin” if the received header is ‘0x0CDA0ADC0CDA**0ADC**’



File name	: <File_Prefix>_<Raw/Header Mode>_LogFile.csv
File extension	: .csv
Type	: Comma separated text
Description	: Contains the following information

- Record process configuration
  - Log mode
  - LVDS lane mode
  - Record stop mode
  - Maximum captured data file size
- If Post processing, record process summary

- Out of sequence count
- First packet ID
- Last packet ID
- Number of received packets
- Capture start time
- Capture end time
- Capture duration
- If Inline processing, for each data port
  - Dropped packet offset (if any)
  - Number of dropped bytes at the offset (if any)
- If Inline processing, record process summary
  - Out of sequence count
  - Latest out of sequence between <seq num> and <seq num>
  - First packet ID
  - Last packet ID
  - Number of received packets
  - Number of zero filled packets
  - Number of zero filled bytes
  - Capture start time
  - Capture end time
  - Capture duration



```

inline_Raw_LogFile.csv - Notepad
File Edit Format View Help
Start record configuration :
,
Log mode : Raw
LVDS lane mode : 4 lane
Record stop mode : Infinite
Max file size (MB) : 1024,
,*DT 1,

Raw Data :
Out of sequence count - 0
Out of sequence seen from 0 to 0
First Packet ID - 0
Last Packet ID - 0
Number of received packets - 0
Number of zero filled packets - 0
Number of zero filled bytes - 0
Capture start time - Mon Mar 04 09:15:54 2019
Capture end time - Mon Mar 04 09:15:54 2019
Duration(sec) - 0
  
```

## 2.2.6 System Status Codes

Following system status codes will be updated in the shared memory by CLI Record tool which can be read by 'query\_status' command by CLI Control tool.

DCA1000EVM System Status	Bit position	Status Type	Description
STS_NO_LVDS_DATA	0	Warning	If there is no LVDS data from RADAR EVM, after configured timeout seconds CLI would display “No LVDS data”. User must ensure the proper DCA1000EVM FPGA configuration and can restart the capture.
STS_NO_HEADER	1	Warning	If data logging mode is configured to multi-mode and there is no LVDS data from RADAR EVM, after configured timeout seconds CLI would display as “No Header”. User must ensure the proper config file is executed and can restart the capture.
STS_EEPROM_FAILURE	2	Warning	If EEPROM failure happened, CLI would display “EEPROM Failure”. User can power cycle DCA1000EVM and can check EEPROM connectivity and address lines on hardware
STS_SD_CARD_DETECTED	3	NA	NA
STS_SD_CARD_REMOVED	4	NA	NA
STS_SD_CARD_FULL	5	NA	NA
STS_MODE_CONFIG_FAILURE	6	NA	NA
STS_DDR_FULL	7	Warning	If DDR is full, CLI would display “DDR full”. Once DDR is full, DCA1000EVM will truncate the overflown data from RADAR EVM. User can restart the capture with lesser LVDS rate config than the Ethernet data rate and/or lesser delay between record packets.
STS_REC_COMPLETED	8	Warning	If record is completed, CLI would display “Record is completed”. User can ensure the record is stopped after receiving record completion status.
STS_LVDS_BUFFER_FULL	9	Warning	If LVDS buffer is full, CLI would display “LVDS buffer full”. Once LVDS buffer is full, DCA1000EVM will truncate the overflown data from RADAR EVM. User can restart the capture with lesser LVDS rate config than the Ethernet data rate and/or lesser delay between record packets.
STS_PLAYBACK_COMPLETED	10	NA	NA
STS_PLAYBACK_OUT_OF_SEQ	11	NA	NA

Not used	12-15		Future Use
DLL Async Status	Bit position	Status Type	Description
STS_REC_PKT_OUT_OF_SEQ	16	Warning	If record packet is in out of sequence, CLI would display "Packet is in out of sequence". User can restart the capture knowing that data loss might happen.
STS_REC_PROC_TIMEOUT	17	Fatal	If record process is timeout, CLI would display "Record process Timeout error". User must ensure the system connectivity.
STS_REC_FILE_CREATION_ERR	18	Fatal	If record file creation is failed, CLI would display "Record file creation failed". User must ensure the free disk space for data capture.
STS_REC_REORDERING_ERR	19	Fatal	If record reordering is failed in inline processing, CLI would display "Record process reordering failed". User must restart the capture since the captured data is not proper for data alignment algorithm. For 4-lane, algorithm needs 16 bytes of data to do alignment. For 2-lane, algorithm needs 8 bytes of data to do alignment. If the number of bytes in the received packet is not the multiple of 16/8, then the data alignment will fail.
STS_INVALID_RESP_PKT_ERR	20	Warning	If the header and footer is not matching in the received packet over the config port, CLI would display as "Invalid packet received". User can power cycle the DCA1000EVM and start sending commands.
STS_REC_INLINE_BUF_ALLOCATION_ERR	21	Fatal	If record inline buffer memory allocation is failed, CLI would display "Inline buffer allocation failed". Received data will be discarded and not be processed. User must check the RAM size of the PC for data capture and can restart the capture.

**Table 4 Asynchronous status codes**



## 2.3 CLI Application Commands

CLI application commands has the following features

- Handles request and response of configuration control commands.
- Initiate CLI Record tool to run and stop the record process.
- Queries the status of the record process.

Calling Convention -

***DCA1000EVM\_CLI\_Control.exe*** <command> [jsonCfgFile] [-q]

<command>: Supports following commands

[jsonCfgFile]: Json format input parameters text file path

[-q]: Quiet mode – No status display in the console

Commands supported by CLI\_Control tool -

- |                    |                               |
|--------------------|-------------------------------|
| ▪ fpga             | Configure FPGA                |
| ▪ eeprom           | Update EEPROM                 |
| ▪ reset_fpga       | Reset FPGA                    |
| ▪ reset_ar_device  | Reset AR Device               |
| ▪ start_record     | Start Record                  |
| ▪ stop_record      | Stop Record                   |
| ▪ record           | Configure Record delay        |
| ▪ dll_version      | Read DLL version              |
| ▪ fpga_version     | Read FPGA version             |
| ▪ cli_version      | Read CLI version              |
| ▪ query_status     | Read status of record process |
| ▪ query_sys_status | DCA1000EVM System aliveness   |
| ▪ -h               | List of commands supported    |

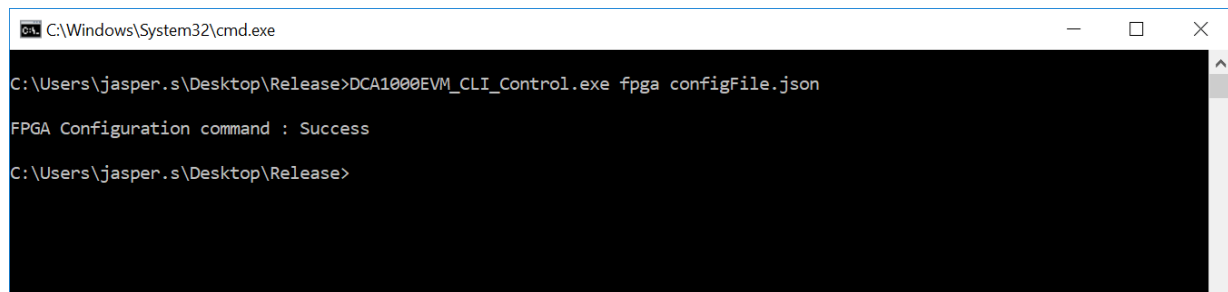
### 2.3.1 Configure FPGA

CLI\_Control tool will use the DLL to send this command to DCA1000EVM over the config port to configure the DCA1000EVM with the following mode configuration

1. Logging Mode (*dataLoggingMode*) – RAW/MULTI
2. LVDS Mode (*lvdsMode*) - 4 lane/2 lane
3. Data Transfer Mode (*dataTransferMode*) – LVDS Capture/Playback
4. Data Capture Mode (*dataCaptureMode*) – SD Card Storage/Ethernet Streaming
5. Data Format Mode (*dataFormatMode*) – 12/14/16 bit
6. Timer (Not configurable using JSON file. Configured by DLL based on the value set in the structure)

Command:

***DCA1000EVM\_CLI\_Control.exe fpga configFile.json***



```

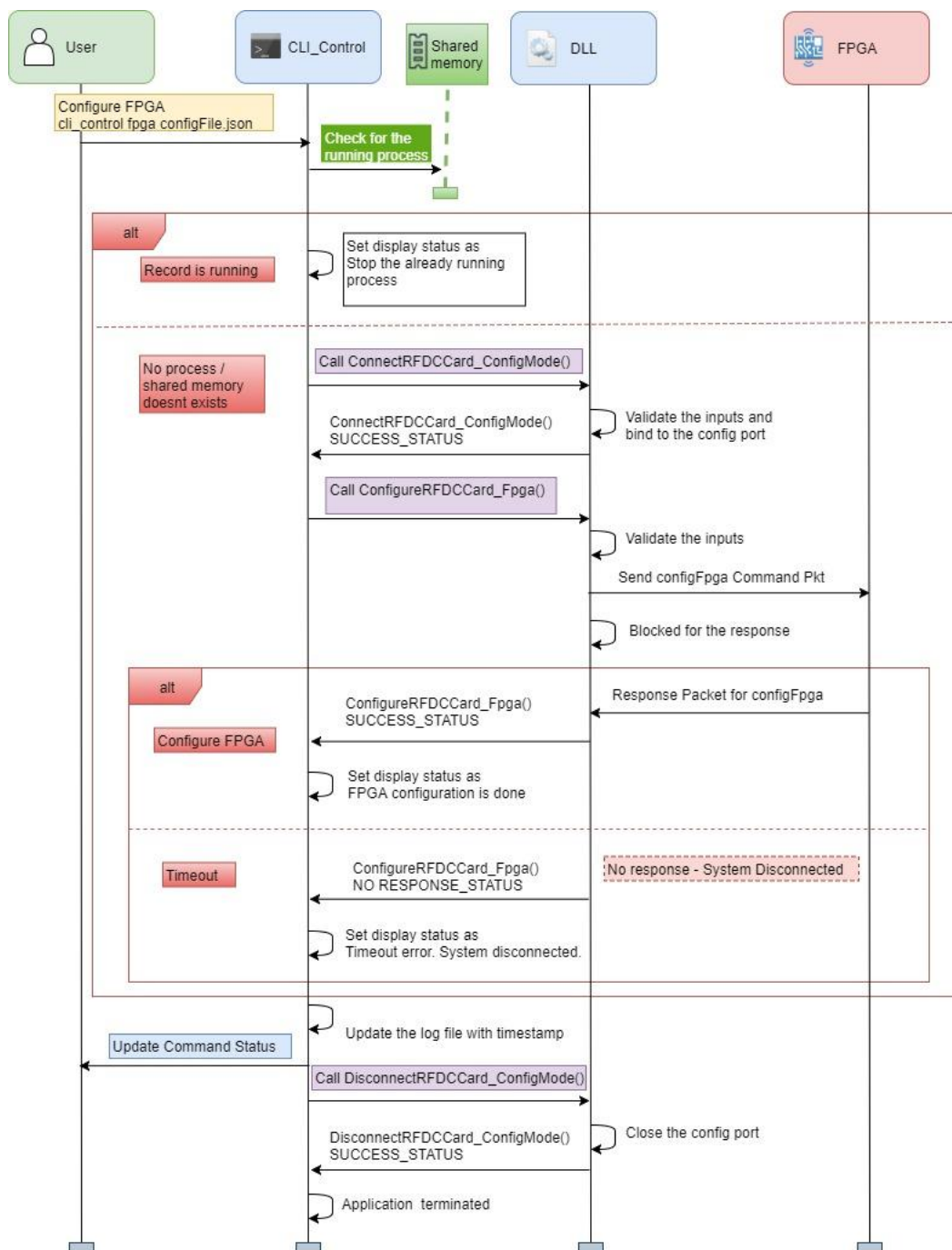
C:\Windows\System32\cmd.exe
C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe fpga configFile.json
FPGA Configuration command : Success
C:\Users\jasper.s\Desktop\Release>
  
```

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
Configure FPGA	<pre> "dataLoggingMode": "raw", "dataTransferMode": "LVDSCapture", "dataCaptureMode": "ethernetStream", "lvdsMode": 1, "dataFormatMode": 3, "ethernetConfig": { "DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096, "DCA1000DataPort": 4098 }           </pre>	<pre> DCA1000EVM_ CLI_Control fpga configFile.json           </pre>	<pre> STATUS ConnectRFDCCard_ConfigMode ( strEthConfigMode sEthConfigMode );  STATUS ConfigureRFDCCard_Fpga ( strFpgaConfigMode sConfigMode );  STATUS DisconnectRFDCCard_ConfigMode (void);           </pre>

**Table 5 Configure FPGA calling convention**

Data Flow Diagram:



**Figure 9 Configure FPGA Data Flow**

Request and Response Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x03	-	-	Command code
Size	UINT16	2	6	-	-	Data size
Data Logging Mode	UINT8	1	1	1	2	1 – Raw mode 2 – Multi mode
LVDS mode	UINT8	1	1	1	2	1 – 4lane 2 – 2lane
Data transfer mode	UINT8	1	1	1	2	1 – LVDS capture 2 – DMM playback
Data capture mode	UINT8	1	2	1	2	1 – SD card storage 2 – Ethernet stream
Data format mode	UINT8	1	2	1	3	1 – 12-bit 2 – 14-bit 3 – 16-bit
Timer	UINT8	1	30	0x0	0xFF	Timer info in seconds.
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet
Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x03	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

**Table 6 Configure FPGA Command Structure**

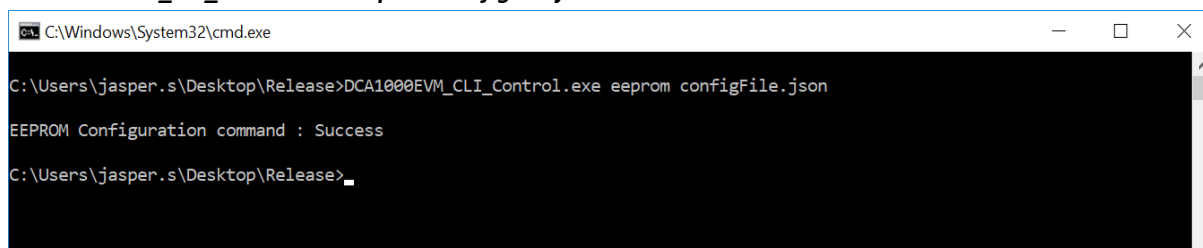
### 2.3.2 Configure EEPROM

CLI\_Control tool will use the DLL to send this command to DCA1000EVM over the config port to reconfigure the IP address of the DCA1000EVM with the following configuration

1. MAC ID (*ethernetConfigUpdate* -> *DCA1000MACAddress*)
2. PC IP Address (*ethernetConfigUpdate* -> *systemIPAddress*)
3. Board IP Address (*ethernetConfigUpdate* -> *DCA1000IPAddress*)
4. Record port number (*ethernetConfigUpdate* -> *DCA1000DataPort*)
5. Configuration port number (*ethernetConfigUpdate* -> *DCA1000ConfigPort*)

Command:

***DCA1000EVM\_CLI\_Control.exe eeprom configFile.json***



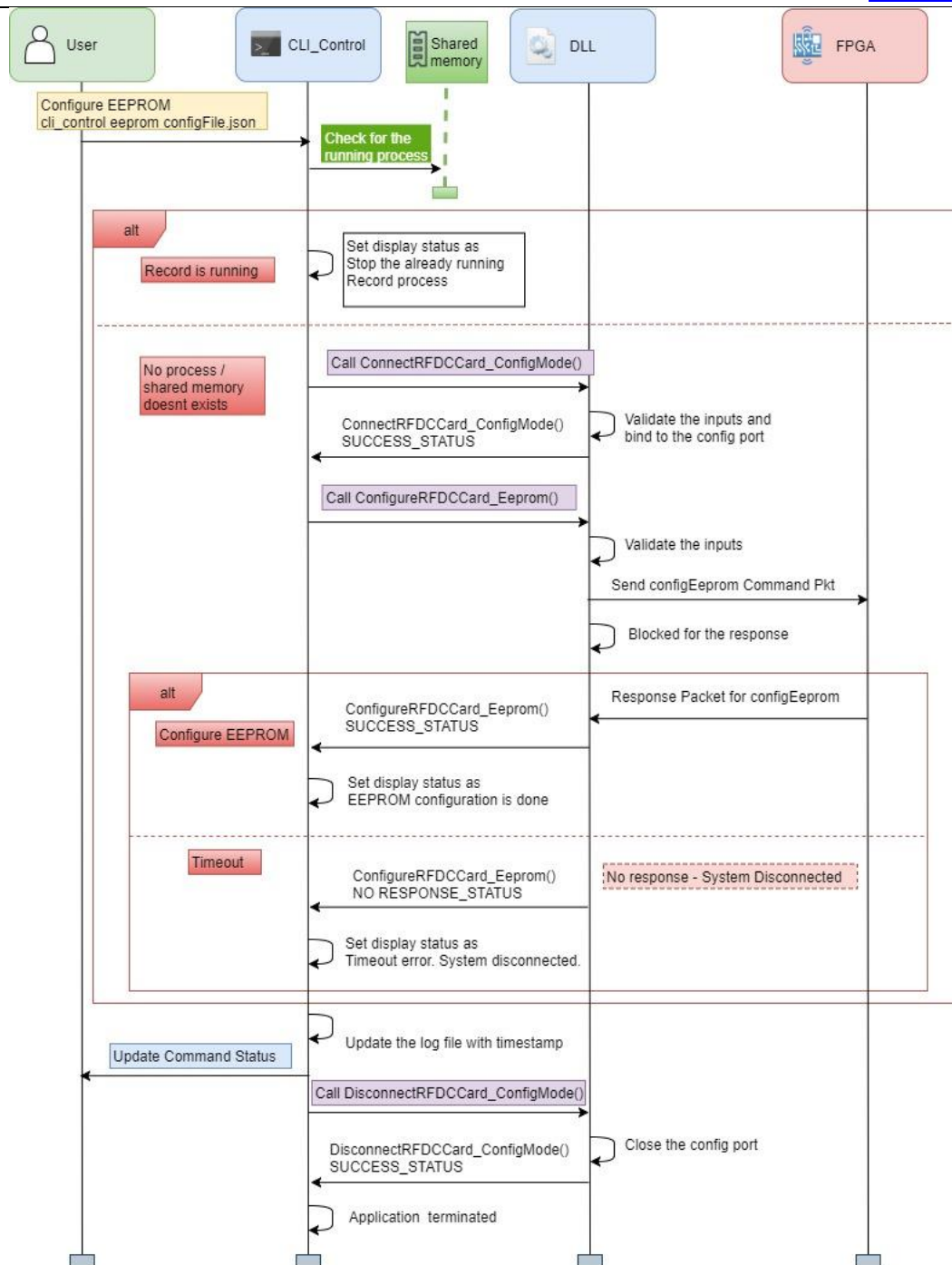
```
C:\Windows\System32\cmd.exe
C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe eeprom configFile.json
EEPROM Configuration command : Success
C:\Users\jasper.s\Desktop\Release>
```

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
Configure EEPROM	<pre>"ethernetConfig": {   "DCA1000IPAddress":     "192.168.33.180",   "DCA1000ConfigPort":     4096,   "DCA1000DataPort": 4098 }, "ethernetConfigUpdate": {   "systemIPAddress":     "192.168.33.30",   "DCA1000IPAddress":     "192.168.33.180",   "DCA1000MACAddress":     "12.34.56.78.90.12",   "DCA1000ConfigPort":     4096,   "DCA1000DataPort": 4098 }</pre>	<pre>DCA1000EVM _CLI_Control eeprom configFile.json</pre>	<pre>STATUS ConnectRFDCCard_ConfigMode (   strEthConfigMode sEthConfigMode );  STATUS ConfigureRFDCCard_Eeprom (   strEthConfigMode sEthConfigMode );  STATUS DisconnectRFDCCard_ConfigMode (void);</pre>

**Table 7 Configure EEPROM calling convention**

Data Flow Diagram:



**Figure 10 Configure EEPROM Data Flow**

Request and Response Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x04	-	-	Command code
Size	UINT16	2	18	-	-	Data size
System IP address	UINT8	4	192.168.33.30	0x0	0xFF	IP address in 4 bytes For example: 0 <sup>th</sup> byte – 30 1 <sup>st</sup> byte – 33 2 <sup>nd</sup> byte – 168 3 <sup>rd</sup> byte – 192
FPGA IP address	UINT8	4	192.168.33.180	0x0	0xFF	IP address in 4 bytes For example: 0 <sup>th</sup> byte – 180 1 <sup>st</sup> byte – 33 2 <sup>nd</sup> byte – 168 3 <sup>rd</sup> byte – 192
FPGA MAC address	UINT8	6	12-34-56-78-90-12	0x0	0xFF	MAC address in 6 bytes For example: 0 <sup>th</sup> byte- 12 1 <sup>st</sup> byte – 90 2 <sup>nd</sup> byte – 78 3 <sup>rd</sup> byte – 56 4 <sup>th</sup> byte – 34 5 <sup>th</sup> byte – 12
Configuration port	UINT16	2	4096	0x1	0xFFFF	Config Port number
Data port	UINT16	2	4098	0x1	0xFFFF	Data Port number
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet
Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x04	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success

						1 – Failure
<b>Footer</b>	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

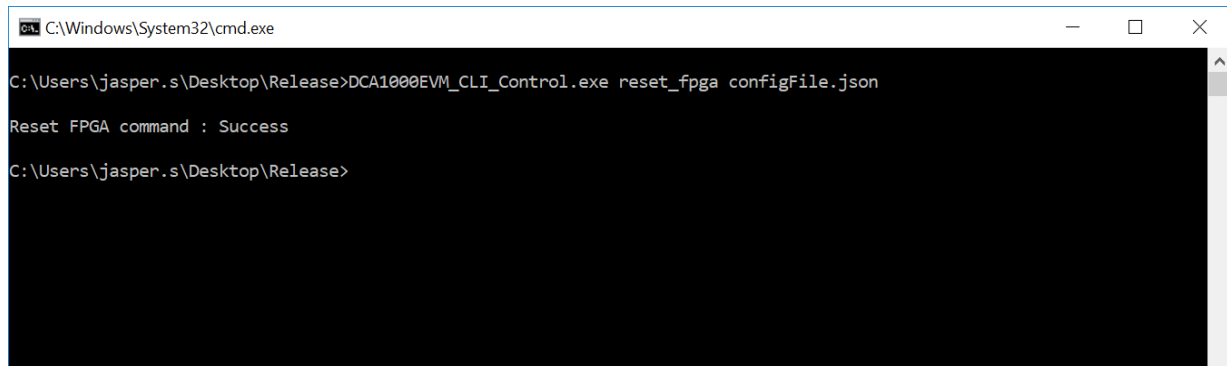
**Table 8 Configure EEPROM Command Structure**

### 2.3.3 Reset FPGA

CLI\_Control tool will use the DLL to send this command to DCA1000EVM over the config port to reset the DCA1000EVM FPGA.

Command:

***DCA1000EVM\_CLI\_Control.exe reset\_fpga configFile.json***



```

C:\Windows\System32\cmd.exe

C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe reset_fpga configFile.json

Reset FPGA command : Success

C:\Users\jasper.s\Desktop\Release>

```

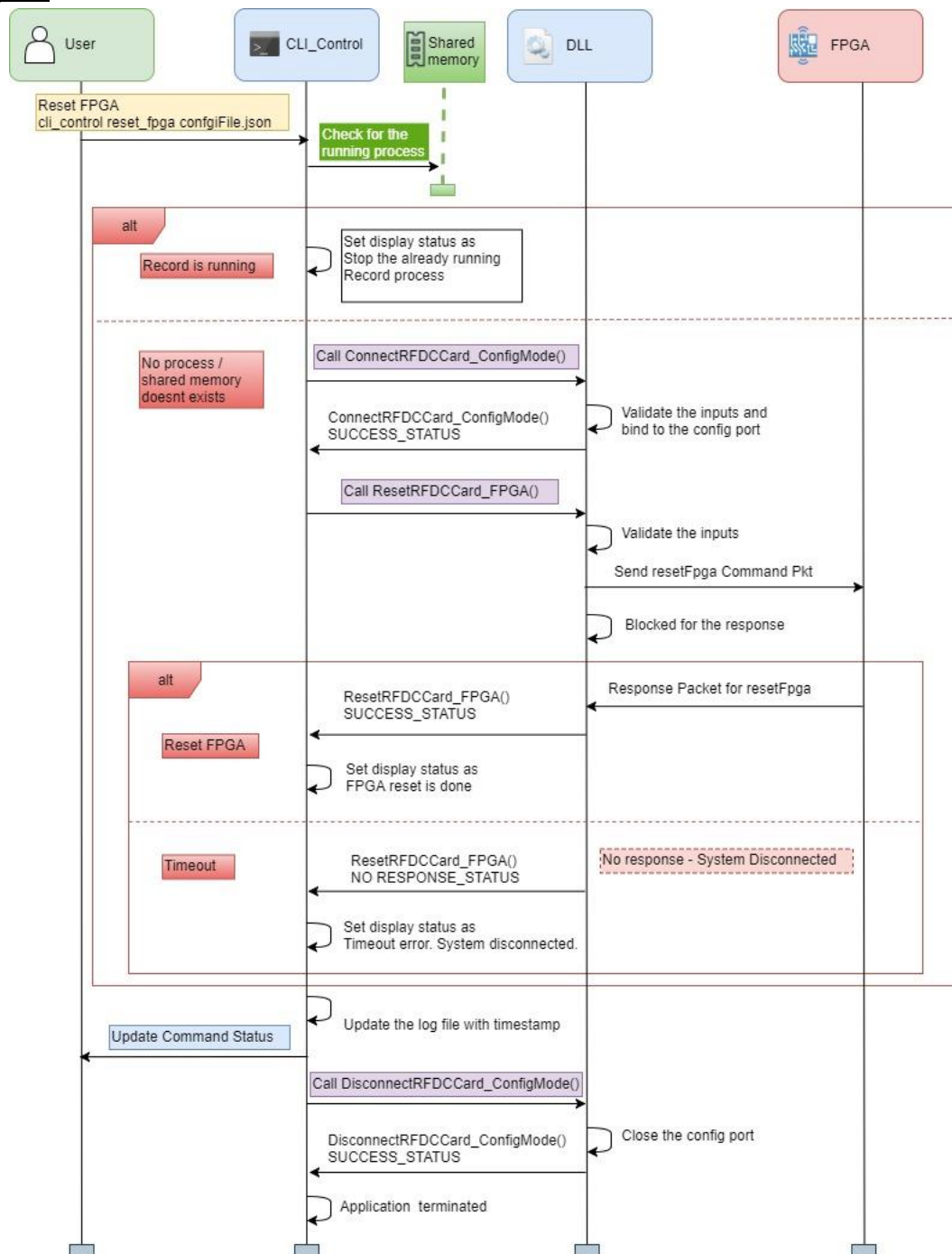
JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
Reset FPGA	"ethernetConfig": { "DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096, "DCA1000DataPort": 4098 }	DCA1000EVM _CLI_Control reset_fpga configFile.json	STATUS ConnectRFDCCard_ConfigMode ( strEthConfigMode sEthConfigMode );  STATUS ResetRFDCCard_FPGA (void);  STATUS DisconnectRFDCCard_ConfigMode (void);

**Table 9 Reset FPGA calling convention**



Data Flow Diagram:



**Figure 11 Reset FPGA Data Flow**

Request and Response Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x01	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet
Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x01	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

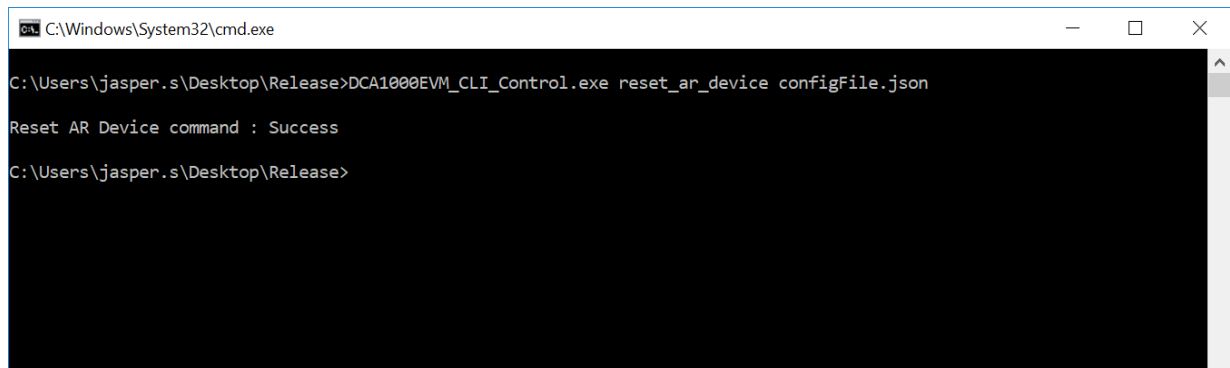
**Table 10 Reset FPGA Command Structure**

### 2.3.4 Reset RADAR EVM

CLI\_Control tool will use the DLL to send this command to DCA1000EVM over the config port to reset RADAR EVM.

Command:

***DCA1000EVM\_CLI\_Control.exe reset\_ar\_device configFile.json***



```

C:\Windows\System32\cmd.exe

C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe reset_ar_device configFile.json
Reset AR Device command : Success
C:\Users\jasper.s\Desktop\Release>

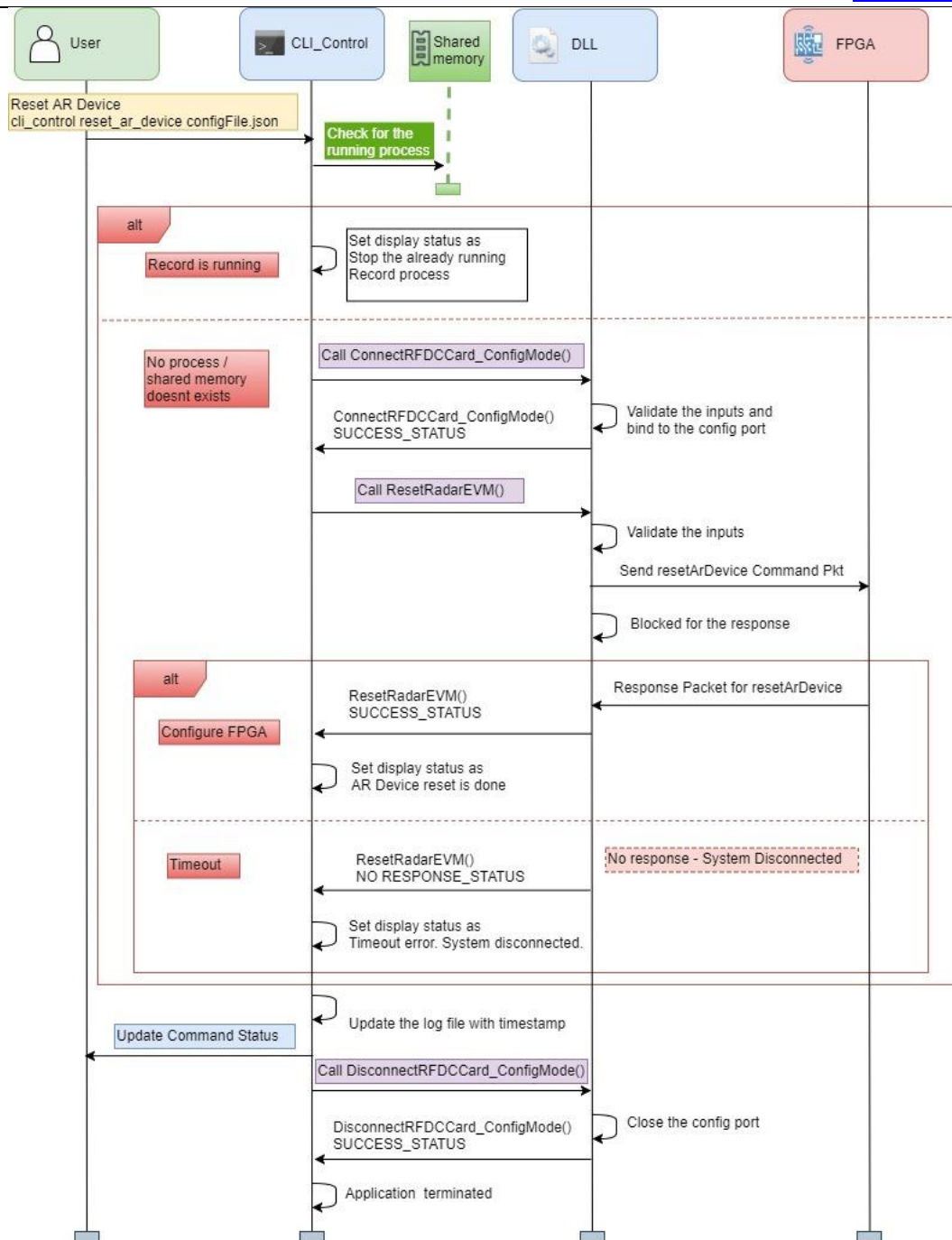
```

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
Reset RADAR EVM	"ethernetConfig": { "DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096, "DCA1000DataPort": 4098 }	DCA1000EVM_ CLI_Control reset_ar_device configFile.json	STATUS ConnectRFDCCard_ConfigMode ( strEthConfigMode sEthConfigMode );  STATUS ResetRadarEVM (void);  STATUS DisconnectRFDCCard_ConfigMode (void);

**Table 11 Reset RADAR EVM calling convention**

Data Flow Diagram:



**Figure 12 Reset RADAR EVM Data Flow**

#### Request and Response Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
------	-----------	-----------------	---------------	-----------	-----------	-------------

Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x02	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet
Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x02	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

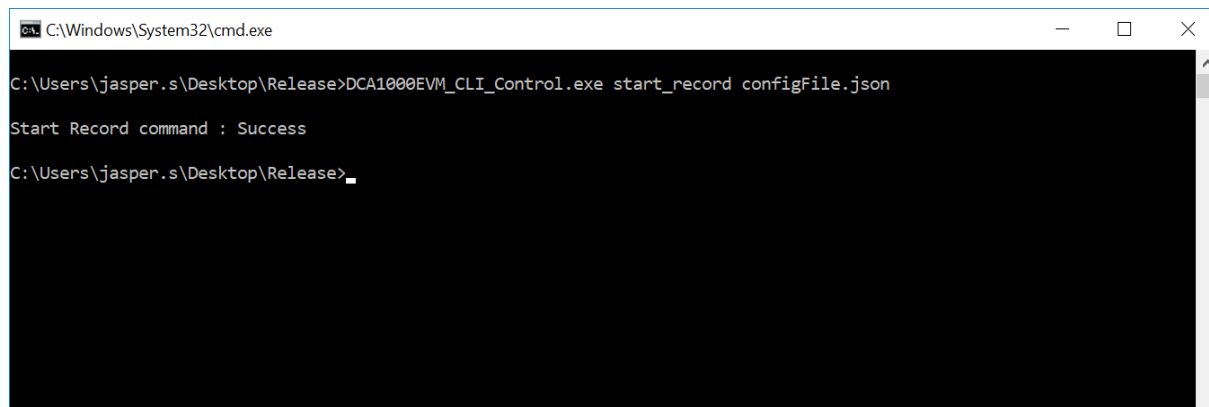
**Table 12 Reset RADAR EVM Command Structure**

### 2.3.5 Start record

CLI\_Control tool will use the DLL to verify the DCA1000EVM connectivity and invoke the CLI Record tool. CLI Record tool will use the DLL to send this command to DCA1000EVM over the config port to start the recording.

Command:

***DCA1000EVM\_CLI\_Control.exe start\_record configFile.json***



```

C:\Windows\System32\cmd.exe
C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe start_record configFile.json
Start Record command : Success
C:\Users\jasper.s\Desktop\Release>

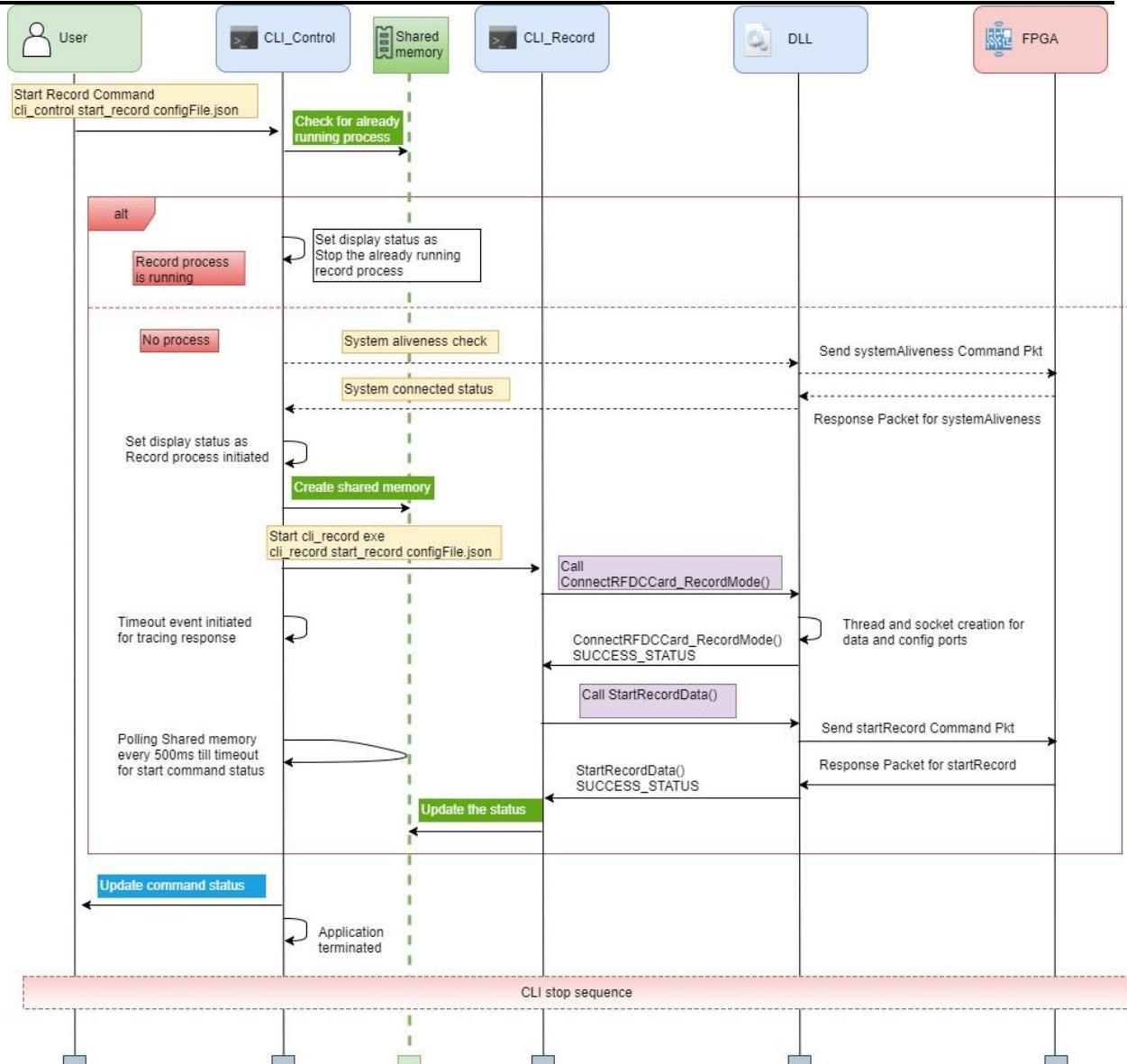
```

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
Start Record	<pre>"dataLoggingMode": "raw", "ethernetConfig": { "DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096, "DCA1000DataPort": 4098 }, "captureConfig": { "fileBasePath": "C:\\Users\\CLI_Inline", "filePrefix": "inline", "maxRecFileSize_MB": 1024, "sequenceNumberEnable": 1, "captureStopMode": "infinite", "bytesToCapture": 4000, "durationToCapture_ms": 4000, "framesToCapture": 40 }, "dataFormatConfig": { "MSBToggle": 0, "reorderEnable": 1,</pre>	<pre>DCA1000EV M_CLI_Cont rol start_record configFile.js on</pre>	<pre>STATUS ConnectRFDCCard_RecordMode ( strEthConfigMode sEthConfigMode );  STATUS StartRecordData ( strStartRecConfigMode sStartRecConfigMode );  STATUS DisconnectRFDCCard_RecordMode (void);</pre>

**Table 13 Start Record calling convention**

Data Flow Diagram:



**Figure 13 Start Record Data Flow**

Request and Response Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x05	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always.

						Stop bits of packet
Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x05	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

**Table 14 Start Record Command Structure**

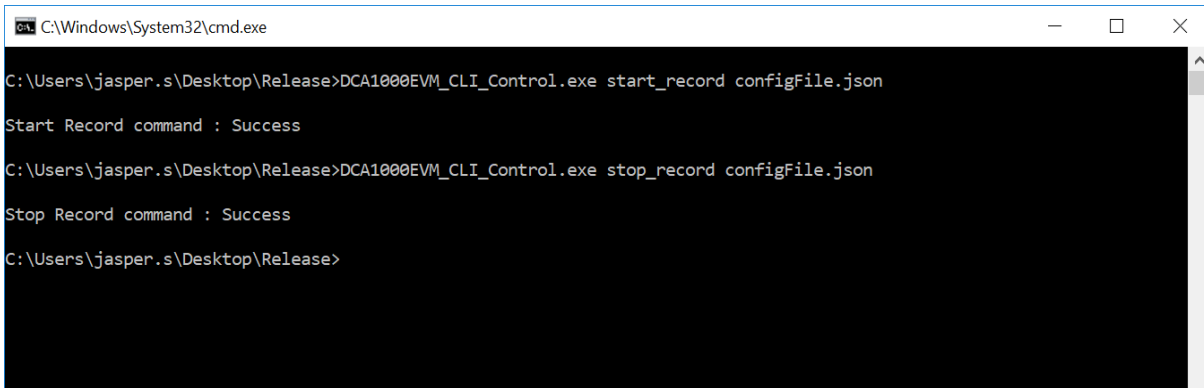
## 2.3.6 Stop record

### 2.3.6.1 User asynchronous stop record

CLI\_Control tool will use DLL to pass the stop message to CLI\_Record tool to stop the recording. On receiving the message, CLI\_Record tool will use the DLL to send this command to DCA1000EVM over the config port to stop the record data streamed over Ethernet from the DCA1000EVM.

Command:

***DCA1000EVM\_CLI\_Control.exe stop\_record configFile.json***



```

C:\Windows\System32\cmd.exe

C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe start_record configFile.json
Start Record command : Success

C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe stop_record configFile.json
Stop Record command : Success

C:\Users\jasper.s\Desktop\Release>

```

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
---------	----------------	------------------------	----------------------------



Stop Record (Based on user stop)	"ethernetConfig": { "DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096, "DCA1000DataPort": 4098 }	DCA1000EVM_ CLI_Control stop_record configFile.json	STATUS ConnectRFDCCard_AsyncCommand Mode ( strEthConfigMode sEthConfigMode );  STATUS StopRecordAsyncCmd (void);  STATUS DisconnectRFDCCard_ AsyncCommandMode (void);
--	--	--	--

**Table 15 User asynchronous Stop Record calling convention**

Data Flow Diagram:

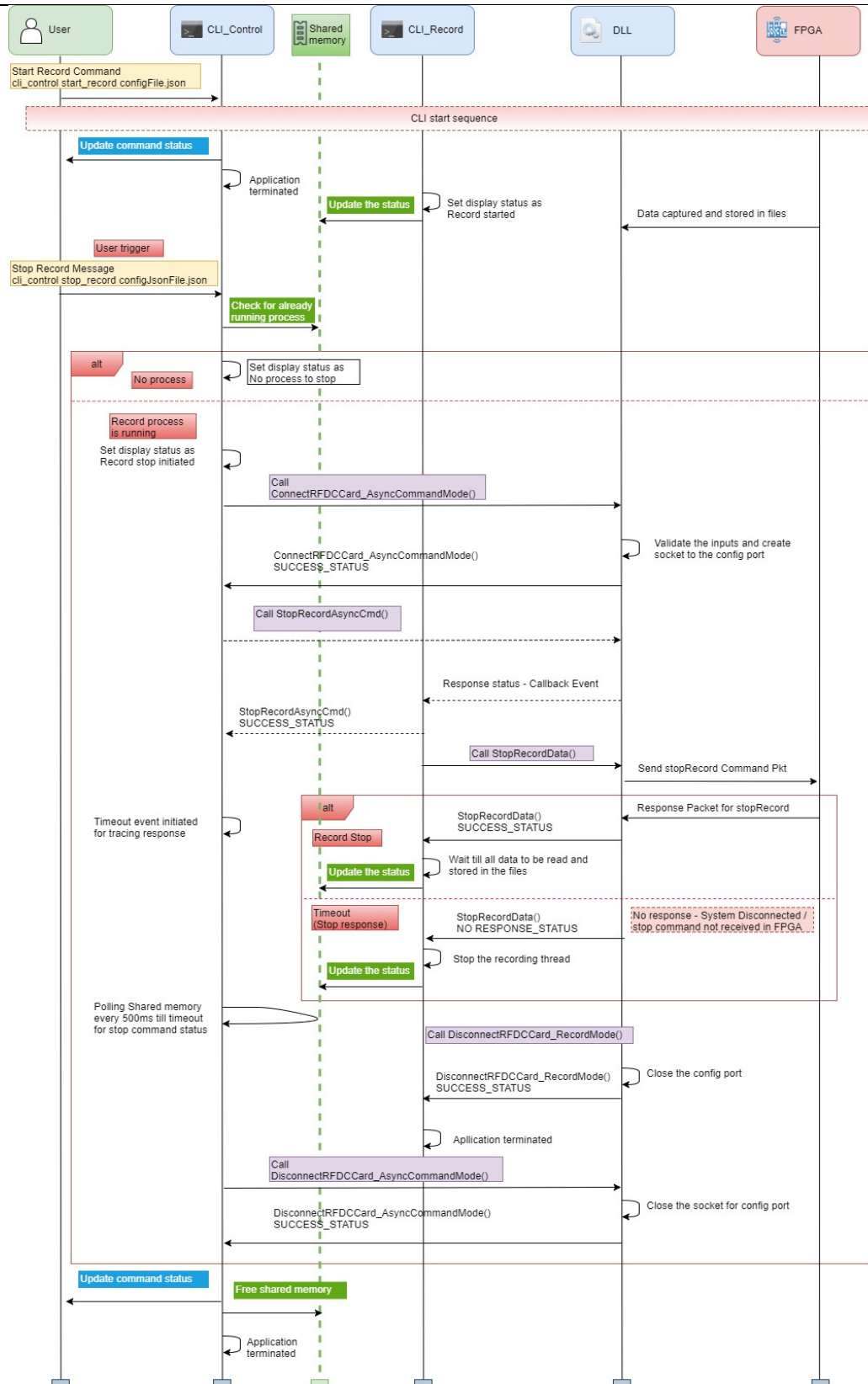


Figure 14 User asynchronous Stop Record Data Flow

Request Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x24	-	-	Command code
Size	UINT16	2	0	-	-	Data size
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

**Table 16 User asynchronous Stop Record Command Structure**

### 2.3.6.2 DCA1000EVM asynchronous status stop record

Once the record completed status is received from DCA1000EVM, CLI\_Record tool will use DLL to send the stop command over the config port to DCA1000EVM and handle its response.

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
Stop Record (Based on DCA1000EVM record completed status)	"ethernetConfig": { "DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096, "DCA1000DataPort": 4098 }	NA	STATUS ConnectRFDCCard_RecordMode ( strEthConfigMode sEthConfigMode );  STATUS StopRecordData (void);  STATUS DisconnectRFDCCard_ RecordMode (void);

Data Flow Diagram:

## DCA1000EVM Architecture Overview

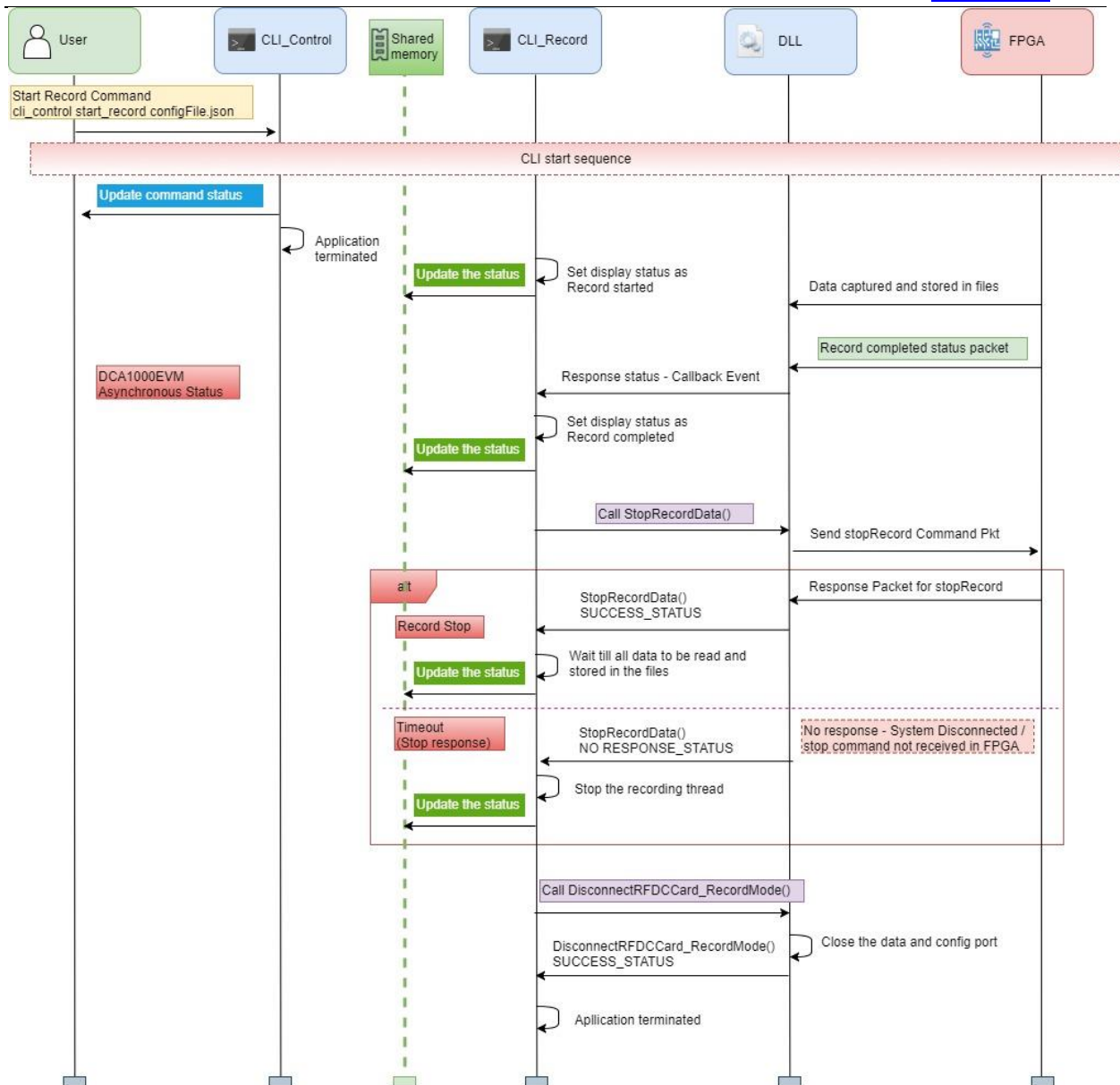
[www.ti.com](http://www.ti.com)


Figure 15 DCA1000EVM asynchronous status Stop Record Data Flow

### Request and Response Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x06	-	-	Command code
Size	UINT16	2	0	-	-	Data size

Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet
Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x06	-	-	Command code
Status	UINT16	2	0	0	1	0 – Success 1 – Failure
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

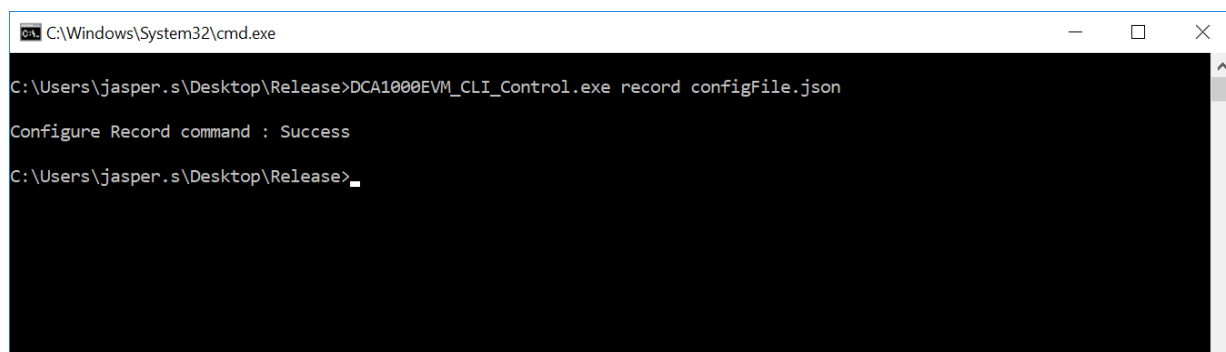
**Table 17 DCA1000EVM asynchronous status Stop Record Command Structure**

### 2.3.7 Configure record delay

CLI\_Control tool will use the DLL to send this command to DCA1000EVM over the config port to configure the delay between record packets.

Command:

***DCA1000EVM\_CLI\_Control.exe record configFile.json***



```

C:\Windows\System32\cmd.exe
C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe record configFile.json
Configure Record command : Success
C:\Users\jasper.s\Desktop\Release>_

```

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
Configure Record delay	"packetDelay_us": 5, "ethernetConfig": { "DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096,	DCA1000EVM_ CLI_Control record configFile.json	STATUS ConnectRFDCCard_ConfigMode ( strEthConfigMode sEthConfigMode );

	"DCA1000DataPort": 4098 }		STATUS ConfigureRFDCCard_Record ( strRecConfigMode sRecConfigMode );  STATUS DisconnectRFDCCard_ConfigMode (void);
--	------------------------------	--	---

**Table 18 Configure record delay calling convention**

Data Flow Diagram:

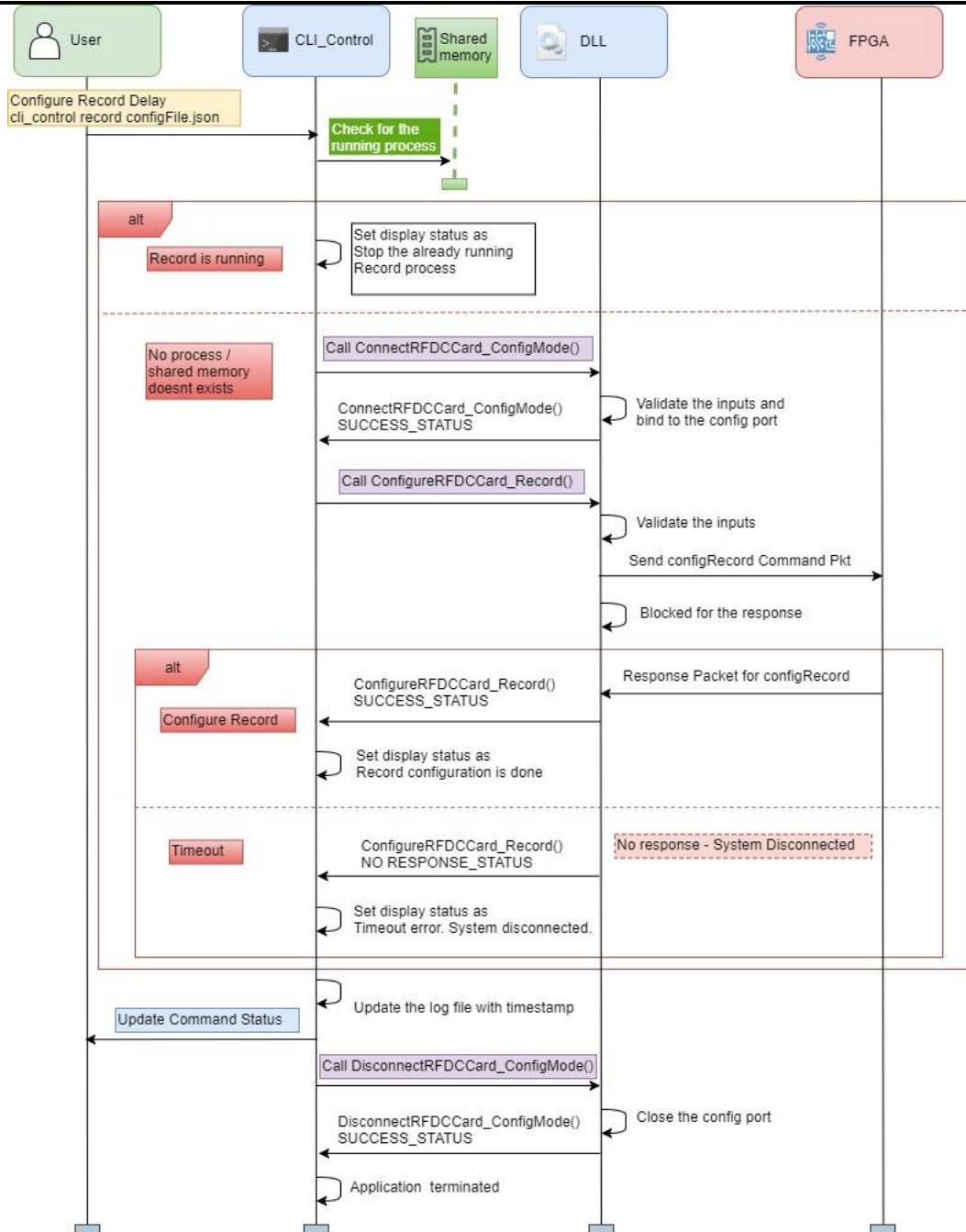


Figure 16 Configure Record Delay Data Flow

Request and Response Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x0B	-	-	Command code

<b>Size</b>	UINT16	2	6	-	-	Data size
<b>Packet Size</b>	UINT16	2	1472	48	1472	Packet size
<b>Delay</b>	UINT16	2	25	5	500	Inter-packet delay
<b>Future Use</b>	UINT16	2	0	-	-	Future Use
<b>Footer</b>	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet
<b>Name</b>	<b>Data Type</b>	<b>Number of bytes</b>	<b>Default Value</b>	<b>Min value</b>	<b>Max value</b>	<b>Description</b>
<b>Header</b>	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
<b>Command code</b>	UINT16	2	0x0B	-	-	Command code
<b>Status</b>	UINT16	2	0	0	1	0 – Success 1 – Failure
<b>Footer</b>	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

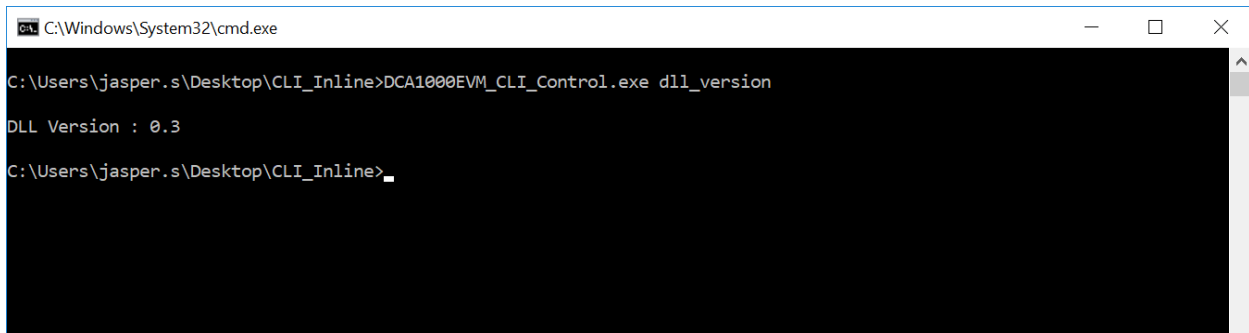
Table 19 Configure record delay command structure

### 2.3.8 Read DLL version

CLI\_Control tool will use DLL to execute this command to read DLL version.

Command:

***DCA1000EVM\_CLI\_Control.exe dll\_version***



```

C:\Windows\System32\cmd.exe
C:\Users\jasper.s\Desktop\CLI_Inline>DCA1000EVM_CLI_Control.exe dll_version
DLL Version : 0.3
C:\Users\jasper.s\Desktop\CLI_Inline>_

```

JSON File data and command calling conventions:

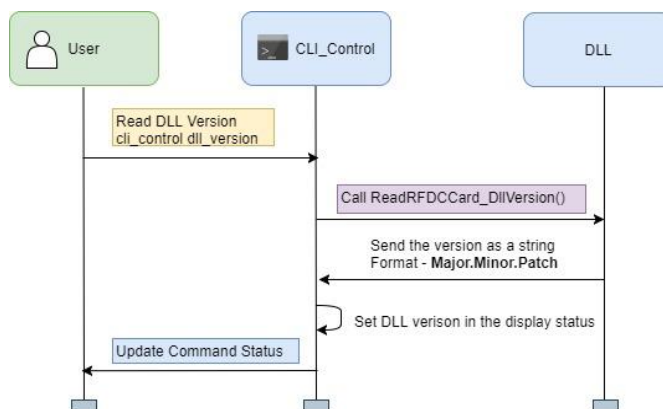
Command	JSON File data	CLI calling convention	DLL API calling convention
---------	----------------	------------------------	----------------------------



Read DLL version	NA	DCA1000EVM_CLI_Control dll_version	STATUS ReadRFDCCard_DllVersion (SINT8 *s8DllVersion);
------------------	----	---------------------------------------	---

**Table 20 Read DLL version calling convention**

Data Flow Diagram:

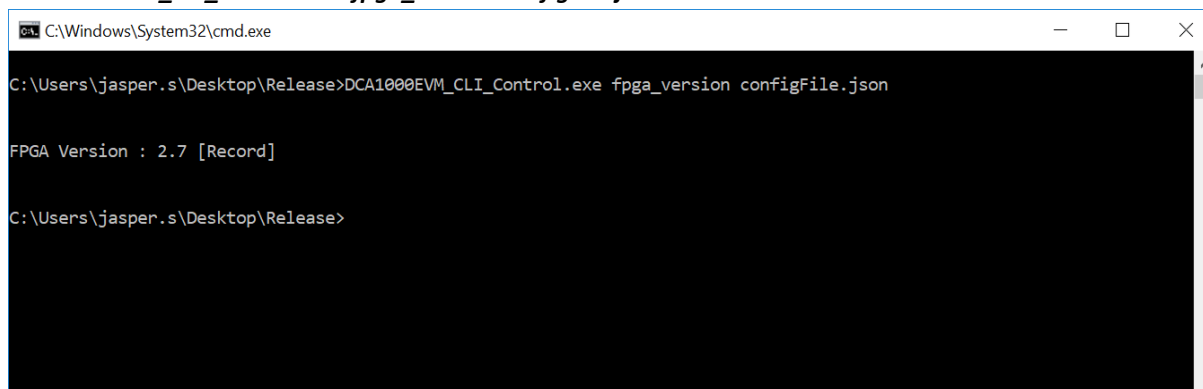

**Figure 17 Read DLL Version Data Flow**

### 2.3.9 Read FPGA version

CLI\_Control tool will use the DLL to send this command to DCA1000EVM over the config port to read FPGA version of DCA1000EVM.

Command:

***DCA1000EVM\_CLI\_Control.exe fpga\_version configFile.json***



```

C:\Windows\System32\cmd.exe

C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe fpga_version configFile.json

FPGA Version : 2.7 [Record]

C:\Users\jasper.s\Desktop\Release>
  
```

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
Read FPGA version	"ethernetConfig": {	DCA1000EVM _CLI_Control	STATUS ConnectRFDCCard_ConfigMode

	"DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096, "DCA1000DataPort": 4098 }	fpga_version configFile.json	<pre>(   strEthConfigMode sEthConfigMode );  STATUS ReadRFDCCard_FpgaVersion (SINT8 *s8FpgaVersion);  STATUS DisconnectRFDCCard_ConfigMode (void);</pre>
--	---	---------------------------------	--

**Table 21 Read FPGA version calling convention**

Data Flow Diagram:

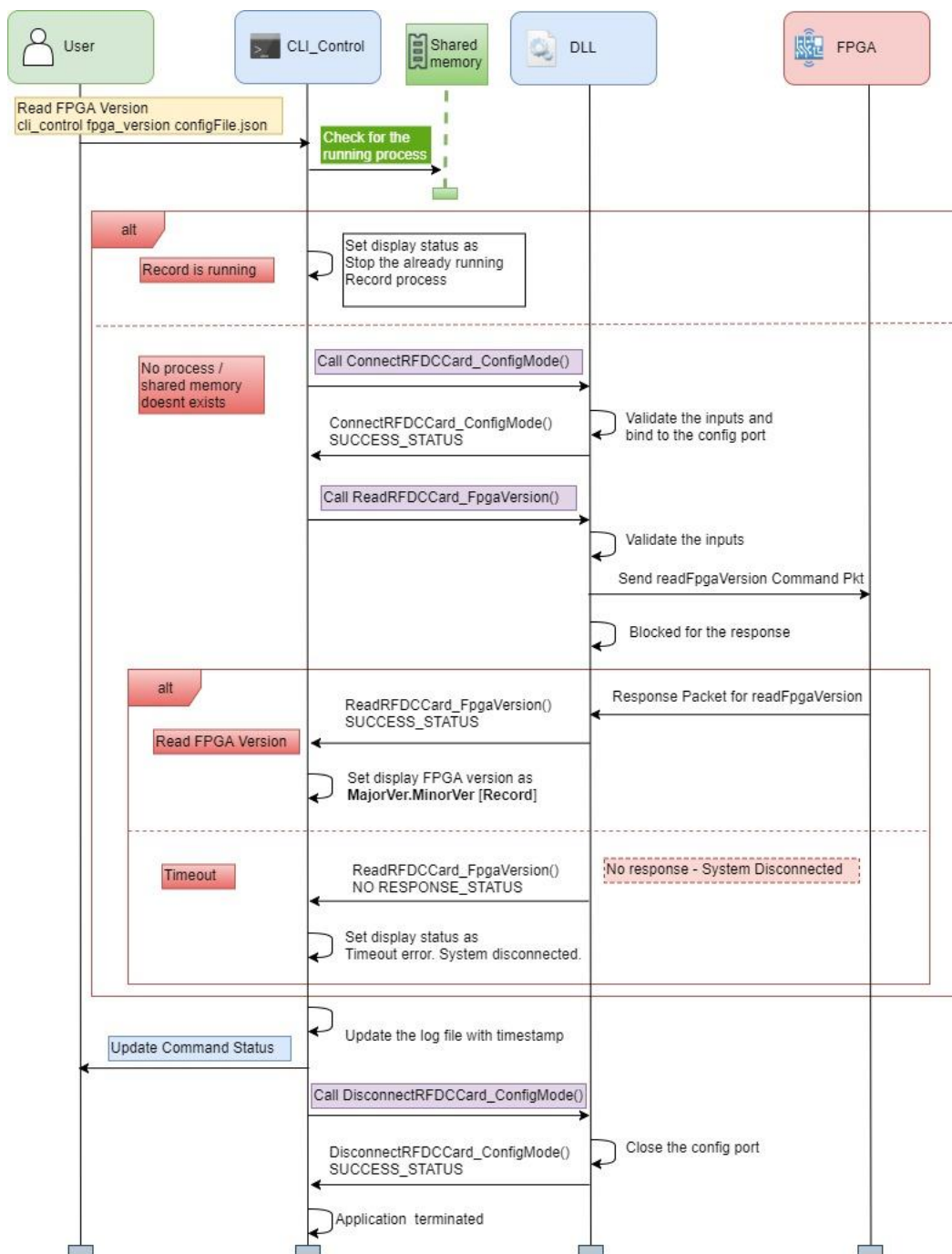


Figure 18 Read FPGA Version Data Flow

Request and Response Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always.

						Start bits of packet
<b>Command code</b>	UINT16	2	0x0E	-	-	Command code
<b>Size</b>	UINT16	2	0	-	-	Data size
<b>Footer</b>	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet
<b>Name</b>	<b>Data Type</b>	<b>Number of bytes</b>	<b>Default Value</b>	<b>Min value</b>	<b>Max value</b>	<b>Description</b>
<b>Header</b>	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
<b>Command code</b>	UINT16	2	0x0E	-	-	Command code
<b>Status</b>	UINT16	2	-	0	0xFFFF	0 – 6 <sup>th</sup> bits -> Major version 7 <sup>th</sup> – 13 <sup>th</sup> bits -> Minor version 14 <sup>th</sup> bit -> 0 – Record bit file 14 <sup>th</sup> bit -> 1 – Playback bit file
<b>Footer</b>	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

**Table 22 Read FPGA version command structure**

### 2.3.10 Query record process status

CLI\_Control tool will execute this command to read the state of the record process and summary of record processing at the command execution point of time. This command can be executed when the CLI record process is running.

Command:

***DCA1000EVM\_CLI\_Control.exe query\_status configFile.json***

```

Select C:\Windows\System32\cmd.exe

C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe start_record configFile.json

Start Record command : Success

C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe query_status configFile.json

Record is in progress. [status -4029]

C:\Users\jasper.s\Desktop\Release>

```

This command will also display in the console and log in the logfile when any of the async status is received while recording is in progress.

Refer section [2.2.3.1](#) for record process states and [2.2.6](#) for system async status details.

```

Select C:\Windows\System32\cmd.exe

C:\Users\jasper.s\Desktop\CLI_Inline>DCA1000EVM_CLI_Control.exe start_record configfile.json

Start Record command : Success

C:\Users\jasper.s\Desktop\CLI_Inline>DCA1000EVM_CLI_Control.exe query_status configfile.json

Record packet out of sequence

Raw Data :
Out of sequence count - 4
First Packet ID - 1
Out of sequence from 56125 to 56202
Last Packet ID - 116010
Number of received packets - 115853
Number of zero filled packets - 157
Number of zero filled bytes - 228592
Capture start time - Tue Mar 05 11:37:15 2019
Capture end time - Tue Mar 05 11:37:21 2019
Capture Duration(sec) - 6

C:\Users\jasper.s\Desktop\CLI_Inline>

```

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
Query record process status	<pre> "ethernetConfig": { "DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096, "DCA1000DataPort": 4098 } </pre>	DCA1000EVM_CLI_Control query_status configFile.json	NA

**Table 23 Query record process status calling convention**

### Data Flow Diagram:

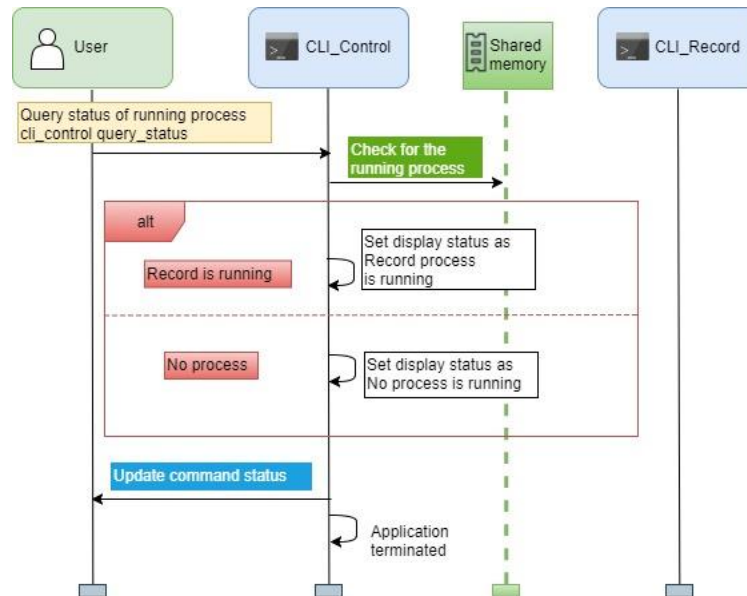


Figure 19 Query Record Process Status Data Flow

### Response Command Format of Asynchronous DCA1000EVM status:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
Command code	UINT16	2	0x0A	-	-	Command code
Status	UINT16	2	0	0	0xFF	Refer <a href="#">2.2.6</a> for system status code
Footer	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

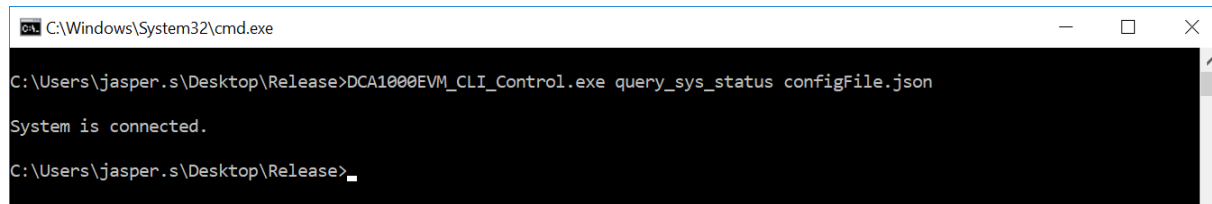
Table 24 Query record process status command structure

### 2.3.11 Query system aliveness status

CLI\_Control tool will use the DLL to send this command to DCA1000EVM over the config port to verify DCA1000EVM system connectivity.

Command:

***DCA1000EVM\_CLI\_Control.exe query\_sys\_status configFile.json***



```

C:\Windows\System32\cmd.exe

C:\Users\jasper.s\Desktop\Release>DCA1000EVM_CLI_Control.exe query_sys_status configFile.json

System is connected.

C:\Users\jasper.s\Desktop\Release>_

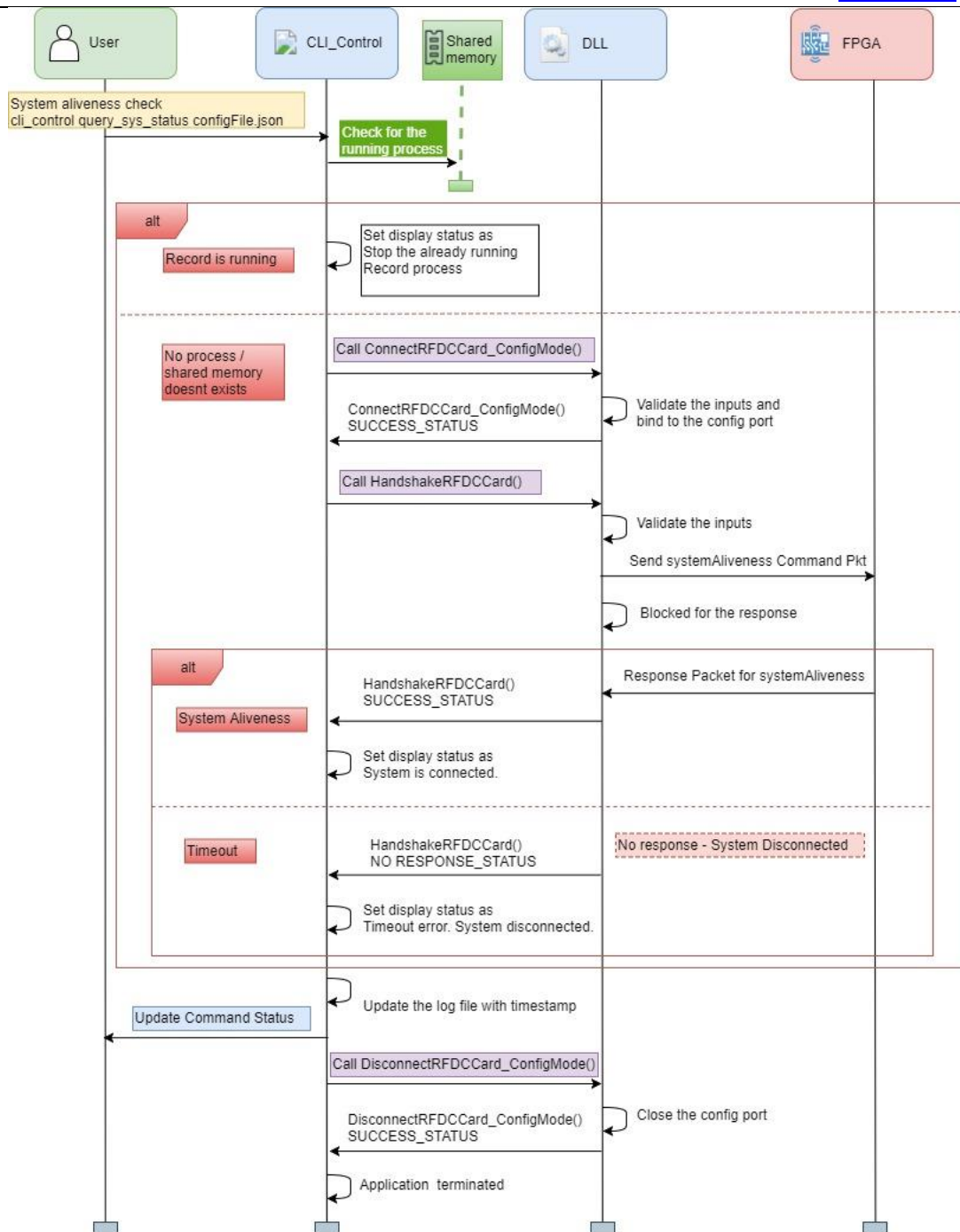
```

JSON File data and command calling conventions:

Command	JSON File data	CLI calling convention	DLL API calling convention
System connectivity	"ethernetConfig": { "DCA1000IPAddress": "192.168.33.180", "DCA1000ConfigPort": 4096, "DCA1000DataPort": 4098 }	DCA1000EVM_CLI_ Control query_sys_status configFile.json	STATUS ConnectRFDCCard_ConfigMode ( strEthConfigMode sEthConfigMode );  STATUS HandshakeRFDCCard(void);  STATUS DisconnectRFDCCard_ConfigMode (void);

**Table 25 Query DCA1000EVM system status calling convention**

Data Flow Diagram:



**Figure 20 Query System Aliveness Data Flow**

#### Request and Response Command Format:

Name	Data Type	Number of bytes	Default Value	Min value	Max value	Description
Header	UINT16	2	0xA55A	-	-	0xA55A always.



						Start bits of packet
<b>Command code</b>	UINT16	2	0x09	-	-	Command code
<b>Size</b>	UINT16	2	0	-	-	Data size
<b>Footer</b>	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet
<b>Name</b>	<b>Data Type</b>	<b>Number of bytes</b>	<b>Default Value</b>	<b>Min value</b>	<b>Max value</b>	<b>Description</b>
<b>Header</b>	UINT16	2	0xA55A	-	-	0xA55A always. Start bits of packet
<b>Command code</b>	UINT16	2	0x09	-	-	Command code
<b>Status</b>	UINT16	2	0	0	1	0 – Success 1 – Failure
<b>Footer</b>	UINT16	2	0xEEAA	-	-	0xEEAA always. Stop bits of packet

**Table 26 System aliveness status Command Structure**

## 2.4 DLL

This section will describe the APIs and inner workings of the DLL. DLL is called by CLI control and record tool internally and an external application interacting with the CLI control doesn't need to call any of these DLL APIs directly. However, an application can choose to skip the CLI control and interact the DLL directly to implement its own way of control. The request and response commands follow defined protocols. The async response of the record commands will be returned to the calling application using callback function.

Application interacting with DLL shall register a callback function for response handling, validation of parameters status and other async status packets. The user event callback with the appropriate error message would be invoked by the API/DLL/Library to communicate the responses, status or error back to the calling application.

### 2.4.1 Commands Protocol

Command protocols used by the DLL to construct the config payload to send to DCA1000EVM over config port are described below.

#### 2.4.1.1 Request packet protocol

Command Request protocol consists of following:

1. Header
  - a. Header
  - b. Command code
  - c. Data size
2. Data
3. Footer

<b>Header</b> (2 bytes)	<b>Command code</b> (2 bytes)	<b>Data size</b> (2 bytes)	<b>Data</b> (Min - 0 bytes Max - 504 bytes)	<b>Footer</b> (2 bytes)
----------------------------	----------------------------------	-------------------------------	---	----------------------------

#### 2.4.1.2 Response packet protocol

Command Response protocol consists of following:

1. Header
  - a. Header
  - b. Command code
2. Status
3. Footer

<b>Header</b> (2 bytes)	<b>Command code</b> (2 bytes)	<b>Status</b> (2 bytes)	<b>Footer</b> (2 bytes)
----------------------------	----------------------------------	----------------------------	----------------------------

#### 2.4.1.3 Data packet protocol

Record data protocol consists of following:

1. DCA1000EVM sends data in the following Ethernet packet format.

<b>Sequence number</b> (4 bytes)	<b>Byte count</b> (6 bytes)	<b>Data</b> (Max – 1456 bytes)
-------------------------------------	--------------------------------	-----------------------------------

2. DLL stores the data in files in the following packet format.

- Sequence number enabled (post processing)

<b>Sequence number</b> (4 bytes)	<b>Data size</b> (4 bytes)	<b>Byte count</b> (6 bytes)	<b>Data</b> (Max – 1456 bytes)
-------------------------------------	-------------------------------	--------------------------------	-----------------------------------

- Sequence number disabled (post processing)

<b>Data</b> (Max – 1456 bytes)
-----------------------------------

- Sequence number enabled/disabled (inline processing)

<b>Data</b> (Max – 1456 bytes)
-----------------------------------

## 2.4.2 Callback Functions

DLL supports callback functions for handling DCA1000EVM asynchronous status and DLL error status. Application that integrate with DCA1000EVM using DLL directly should register to the following two callbacks.

### 2.4.2.1 Asynchronous status callback

CLI Record tool will register its asynchronous callback function using DLL and handles the asynchronous status from DCA1000EVM and DLL errors as following. This callback will be triggered whenever the DCA1000EVM asynchronous status received over the config port.

- DCA1000EVM and DLL async status (Refer [2.2.6](#) for more details)
  - On receiving the status from DLL, CLI Record tool will update the shared memory with the status.
  - For the following 4 status, CLI Record tool will use DLL to send the stop command over the config port to DCA1000EVM and will update the shared memory with the record process state. (Refer section [2.2.3.1](#) for record process states).
    - Record completed
    - No LVDS data
    - No header
    - Reordering of data bytes failure
    - Record file creation failure
  - CLI Record tool will terminate after the stop record API returned the status.

- User asynchronous stop record message
  - On receiving the message from CLI Control tool, CLI\_Record tool will use DLL to send the stop command over the config port to DCA1000EVM and will update the shared memory with the record process state. (Refer section [2.2.3.1](#) for record process states).
  - CLI Record tool will terminate after the stop record API returned the status.
- Record process timeout status
  - On receiving the message from DLL, CLI\_Record tool will use DLL to send the stop command over the config port to DCA1000EVM and will update the shared memory with the record process state. (Refer section [2.2.3.1](#) for record process states).
  - CLI Record tool will terminate after the stop record API returned the status.

Command	JSON File data	CLI calling convention	DLL API calling convention
Asynchronous status	NA	NA	STATUS StatusRFDCCard_EventRegister ( EVENT_HANDLER RFDCCard_EventCallback );

### 2.4.2.2 Record process status callback

CLI Record tool will register its record process status callback function using DLL and update the received record status as following in the shared memory. This callback will be triggered whenever the data packets are received over the data ports.

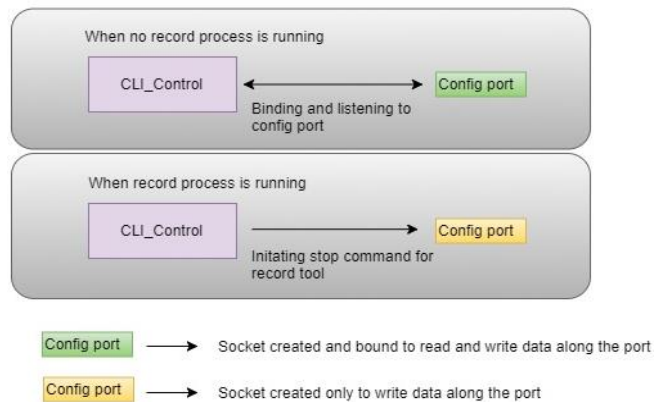
- If Post processing, record process summary for each data port
  - Out of sequence count
  - First packet ID
  - Last packet ID
  - Number of received packets
  - Capture start time
  - Capture end time
  - Capture duration
- If Inline processing, for each data port
  - Dropped packet offset (if any)
  - Number of dropped bytes at the offset (if any)
- If Inline processing, record process summary for each data port
  - Out of sequence count
  - Latest out of sequence between <seq num> and <seq num>
  - First packet ID
  - Last packet ID
  - Number of received packets
  - Number of zero filled packets
  - Number of zero filled bytes

- Capture start time
- Capture end time
- Capture duration

Command	JSON File data	CLI calling convention	DLL API calling convention
Record process status	NA	NA	STATUS RecInlineProcStats_EventRegister ( INLINE_PROC_HANDLER RecordStats_Callback );

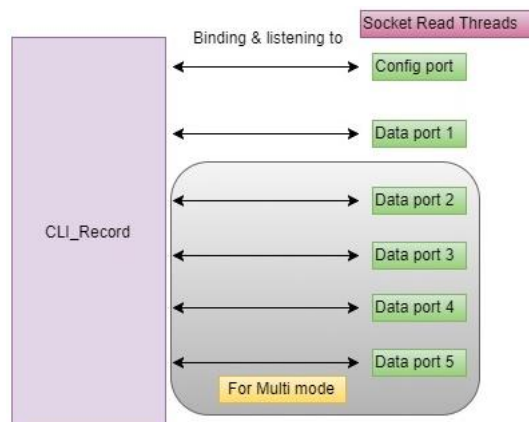
### 2.4.3 Threads

- CLI\_Control tool listens to the command response packets. If any record process is running, only stop and query record process commands are allowed from CLI\_Control tool. CLI\_Control tool reads the status of the record process from shared memory.



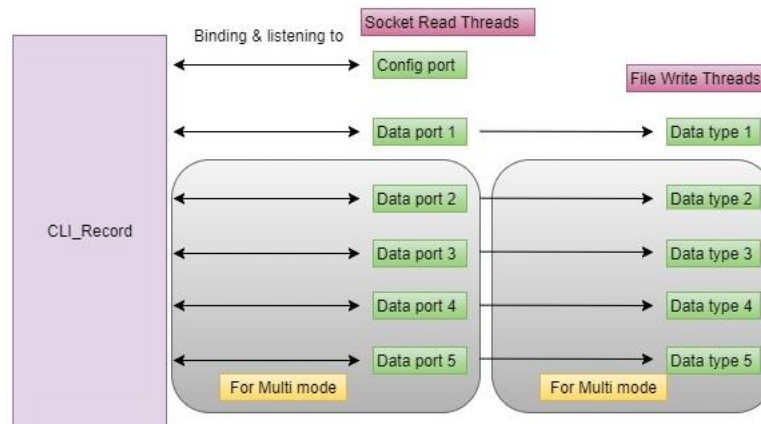
**Figure 21 CLI Control Tool – Thread**

- DLL spawns 2 threads in raw mode data recording and 6 threads in multi-mode data recording when built for post processing mode.



**Figure 22 CLI Record Tool – Post Processing Threads**

- DLL spawns 3 threads in raw mode data recording and 11 threads in multi-mode data recording when built for inline processing mode. Reading data through Ethernet and writing data to the file will be done in parallel.



**Figure 23 CLI Record Tool – Inline Processing Threads**

## 2.4.4 Record data stop mode processing

DLL supports following 4 modes for stop record.

- Bytes
  - DLL will be configured with the *bytes* stop mode and number of bytes to capture, when start record command is sent to DCA1000EVM.
  - On receiving the data packets, the number of bytes will be counted.
  - If any one of the data ports receives the total bytes to be captured, DLL will send the record completed status to the CLI Record tool.
  - CLI Record tool will use the DLL to send the stop record command over the config port through Ethernet.
- Frames
  - DLL will be configured with the *frames* stop mode and number of frames to capture, when start record command is sent to DCA1000EVM.
  - This mode is valid only in multi-mode data capture.
  - On receiving the first packet, the Frame ID will be stored by DLL. The number of frames will be counted if the frame ID is present in the received packet (Every new frame will be received as a new packet and hence frame ID will always be the first 8 bytes of the data payload).
  - If any one of the data ports receives the total frames to be captured, DLL will send the record completed status to the CLI Record tool.
  - CLI Record tool will use the DLL to send the stop record command over the config port through Ethernet.
- Duration
  - DLL will be configured with the *duration* stop mode and duration in millisecond to capture, when start record command is sent to DCA1000EVM.
  - On receiving the first packet, the timer will be started by DLL.

- After the defined duration, DLL will send the record completed status to the CLI Record tool.
  - CLI Record tool will use the DLL to send the stop record command over the config port through Ethernet.
- Infinite
- DLL will be configured with the *infinite* stop mode when start record command is sent to DCA1000EVM.
  - Once the record completed status is received from the DCA1000EVM, DLL will send the record completed status to the CLI Record tool.
  - CLI Record tool will use the DLL to send the stop record command over the config port through Ethernet.

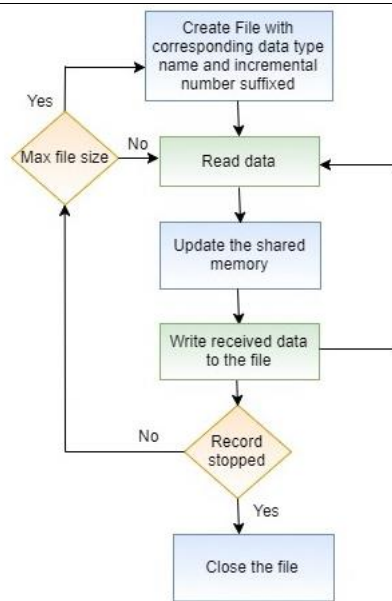
### 2.4.5 Record data processing

DLL application start capturing the data whenever it receives data through Ethernet from DCA1000EVM and stores in the file. Since the data comes as UDP packets, there is a chance of some UDP packets getting dropped or not in sequence. Hence the data should be processed to store in the file in sequence and if any packet is missed, the data should be filled with zeros in the dropped packets.

#### 2.4.5.1 Post processing

For the post processing the following approach has been followed.

- When a packet is received at the data port, the packet is validated against out of order sequence. Following information are getting updated in the shared memory.
- Out of sequence count
  - First packet ID
  - Last packet ID
  - Number of received packets
  - Capture start time
  - Capture end time
  - Capture duration
- Write the packet data into the file and move on to read the next packet till the record process is stopped.



**Figure 24 Post processing - Record to file**

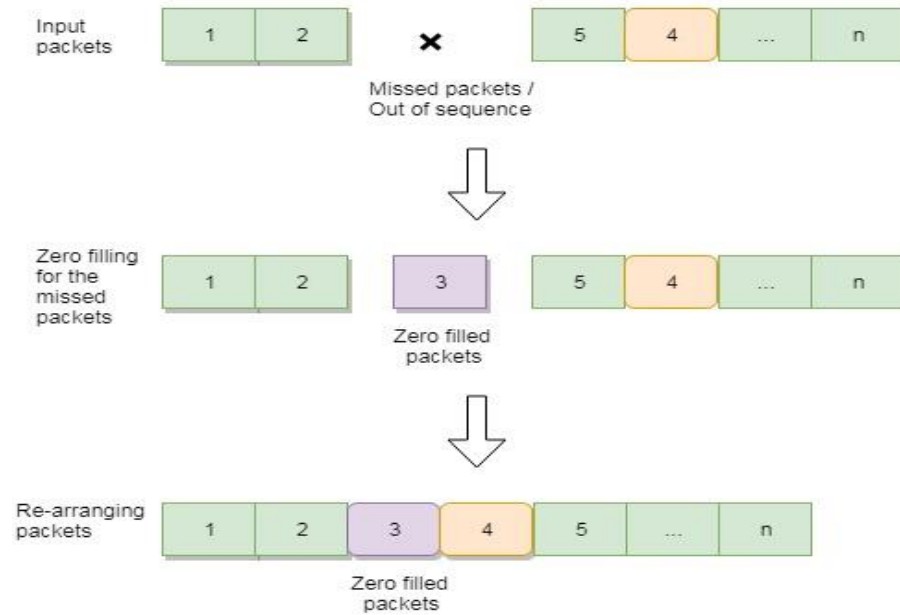
Post processing need following two offline utility to process the captured data.

### 1. Packet Alignment - Offline Utility

In post processing of record process, CLI application captures the data and stores in the files. The stored data need to be processed to fill zeros in the dropped packets if any.

1. Get recorded data filename
2. Store sequence numbers in a buffer
3. Calculate number of packets in the file
4. Adding zeros to the missing packet
5. Packet re-arranging based on sequence number
6. Data realignment happens based on 2or 4 channel data (optional)
7. Output data saved in a file





**Figure 25 Post Processing - Packet Alignment**

## 2. Data Alignment – Offline Utility

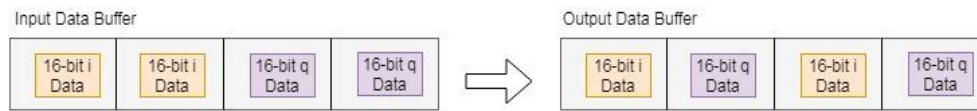
In post processing of record process, CLI application captures the data and stores in the files. The stored data should be re-aligned to make reading of samples easier in post processing.

This utility needs to be run only if captured data is *complex*. It handles only *Format 0* of the LVDS lanes.

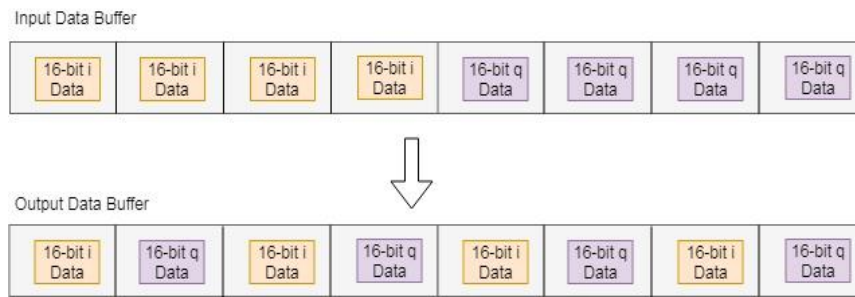
For 2 channel data, the stored data will be like two 16-bit i data followed by two 16-bit q data. These data will be realigned as one 16-bit I data followed by one 16-bit q data and so on.

For 4 channel data, the stored data will be like four 16-bit i data followed by four 16-bit q data. These data will be realigned as one 16-bit I data followed by one 16-bit q data and so on.

#### > 2 Channel Data



#### > 4 Channel Data



**Figure 26 Post Processing - Data Alignment**

### 2.4.5.2 Inline processing

Inline processing or real-time processing of out of order packets can have an impact on the throughput depending on the host PC capabilities. The dropped packet offset and number of bytes dropped in the offset are being captured in the log file after processing the data. For the inline processing the following approach has been followed.

#### **Double Buffer Approach:**

When a packet is received, the packet is validated against out of order sequence.

If the packet is in sequence, write the packet data into the buffer and move on to read the next packet.

If the packet has the sequence number lesser than the expected next packet, seek through the buffer for the packet offset and write the packet data into the buffer. Reset the buffer pointer and move on to read the next packet.

If the packet has the sequence number greater than the expected next packet, calculate the number of packets dropped. Fill the data size with zeros and write the packet data into the buffer and move on to read the next packet.

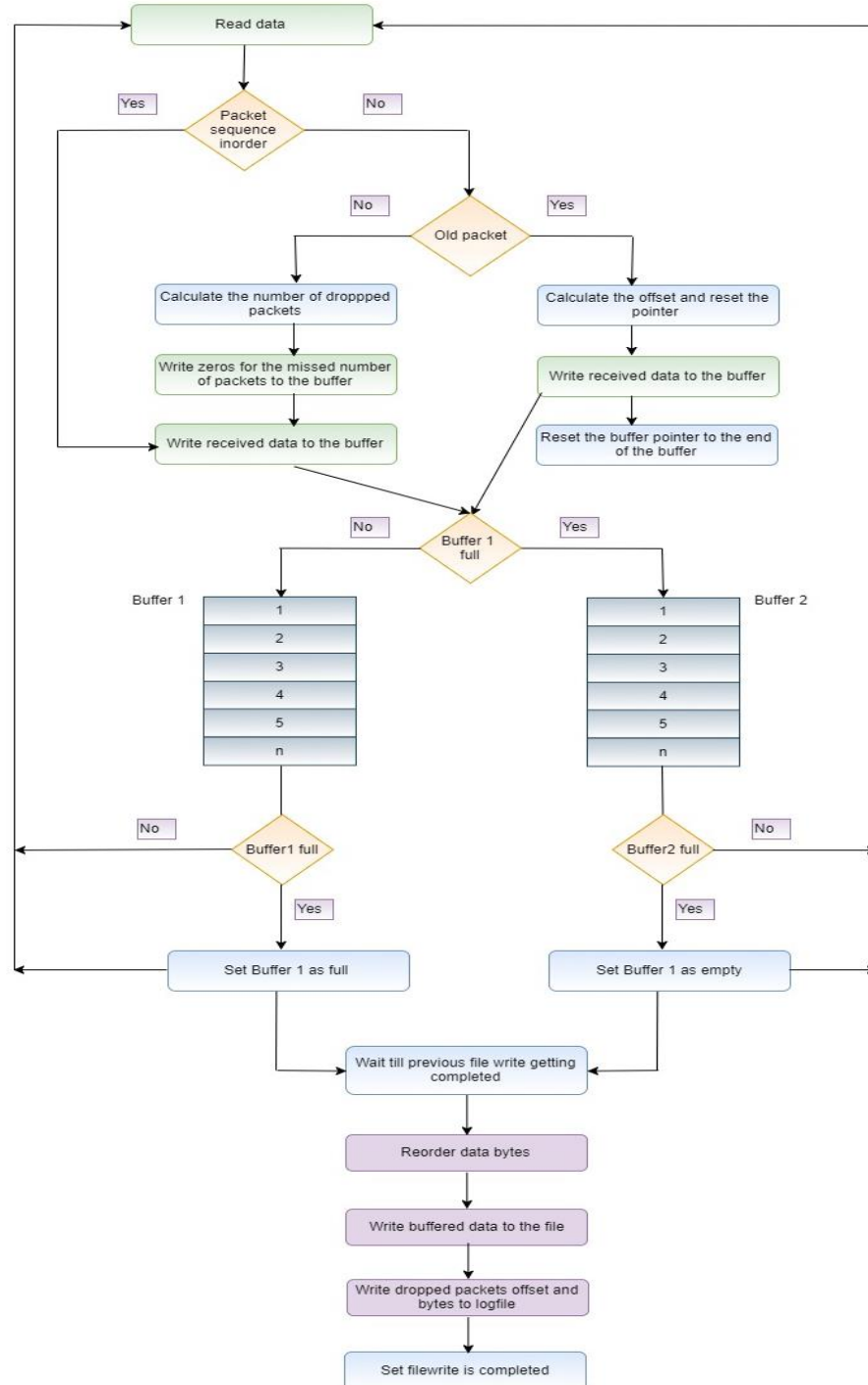
Once the buffer 1 is full, start the file write process using the buffer 1. In parallel, move on to read the packet and fill the received packet in the buffer 2. Once the buffer 2 is full, wait till the previous file write has been completed and start the file write process using the buffer 2. In parallel, move on to read the next packet. Looping the process till record stop is done.

Whenever out of sequence packets encountered, update the log buffer with the dropped packet offset and number of dropped bytes information.

The captured data in the buffer will be re-aligned to make reading of samples easier. After writing the reordered data to the file, update the log file with the log buffer information.

Following information are getting updated in the shared memory.

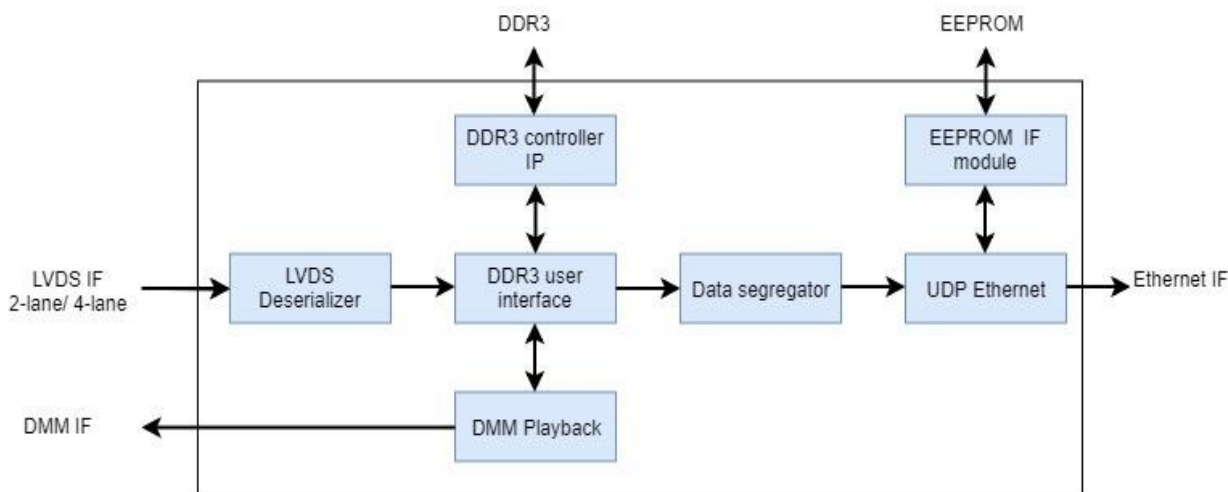
- For each data port
  - Dropped packet offset (if any)
  - Number of dropped bytes at the offset (if any)
- Record process summary
  - Out of sequence count
  - Latest out of sequence between <seq num> and <seq num>
  - First packet ID
  - Last packet ID
  - Number of received packets
  - Number of zero filled packets
  - Number of zero filled bytes
  - Capture start time
  - Capture end time
  - Capture duration



**Figure 27 Inline processing - Record to buffer**

## 2.5 FPGA

The DCA1000EVM DLL encapsulates all the details needed to configure and communicate with the FPGA including reading of the RADAR data over the socket(s). This section will give an overview of the FPGA protocol and high-level implementation for users looking to bypass the DLL and have a need to communicate with the FPGA directly.



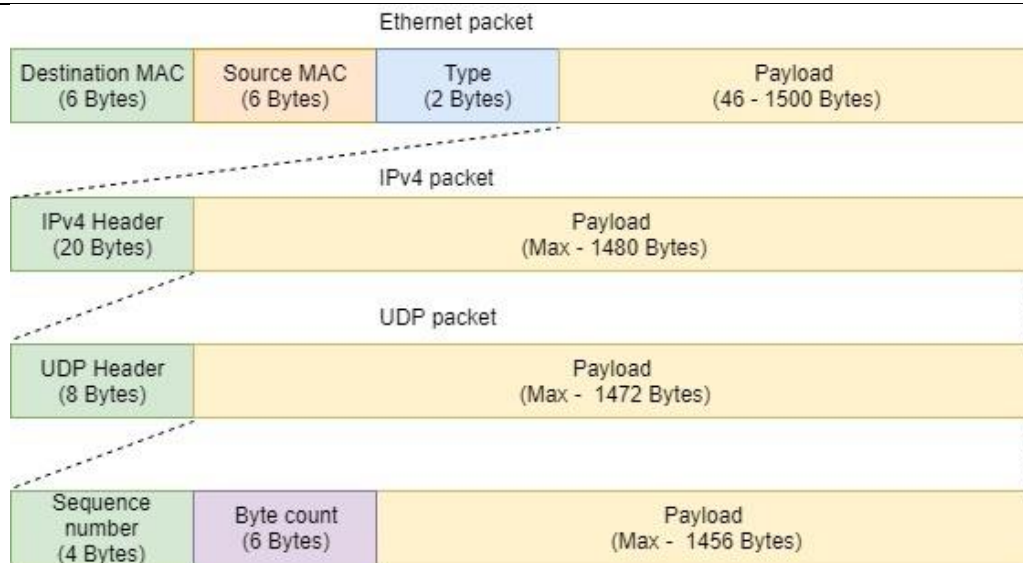
**Figure 28 FPGA high level architecture**

DCA1000EVM system contains Lattice ECP5 FPGA to capture/deserialize LVDS data and forwards to host through Ethernet UDP protocol. Following are the different modules in FPGA data processing.

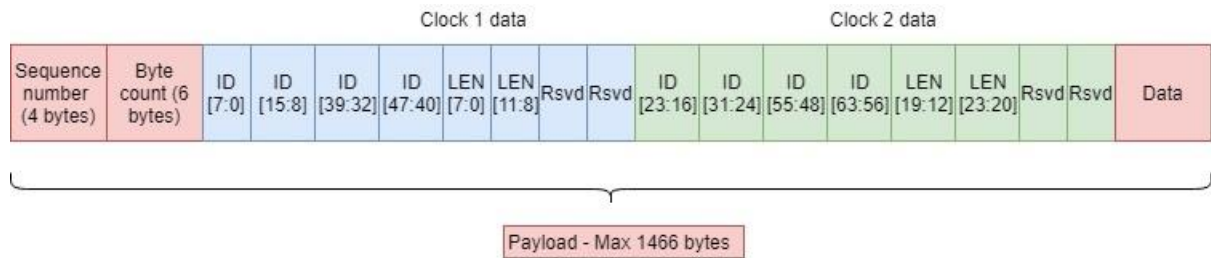
- LVDS module receives 16-bit serial data from TI RADAR EVM and converts it into parallel data.
- DDR3 user interface module commands and controls the DDR3 IP based on data availability from LVDS deserializer module.
- Data segregator module reads data from DDR3 and does following actions.
  - **Raw mode:** Just forwards data to Ethernet through ADC data port.
  - **Multi-mode:** Checks for data type and forwards data to Ethernet through corresponding UDP port.

### Ethernet Module:

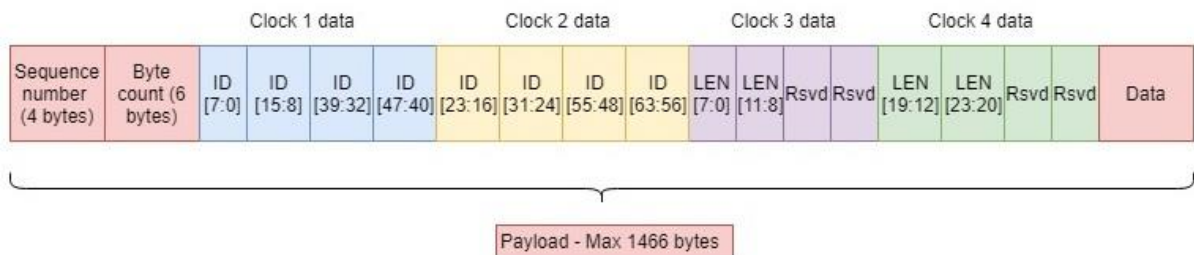
- This module takes the information coming from data segregator module and creates UDP Ethernet packet as shown in figure-29 and forwards to GUI.
- RF Data Capture card sends 4 bytes of sequence number, 6 bytes of byte count and maximum 1456 bytes of payload. Maximum payload size is set to 1456 to maintain I and Q alignment in a packet.



**Figure 29 Ethernet packet format**

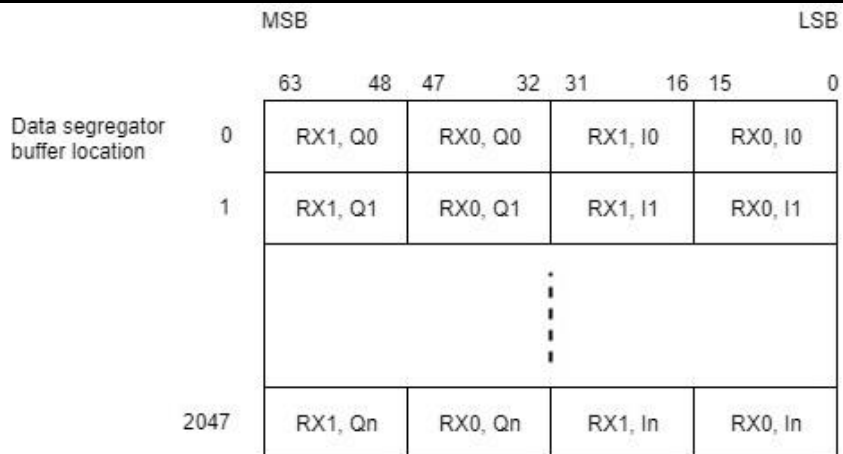


**Figure 30 Ethernet payload format – [Multi mode 4 -lane]**

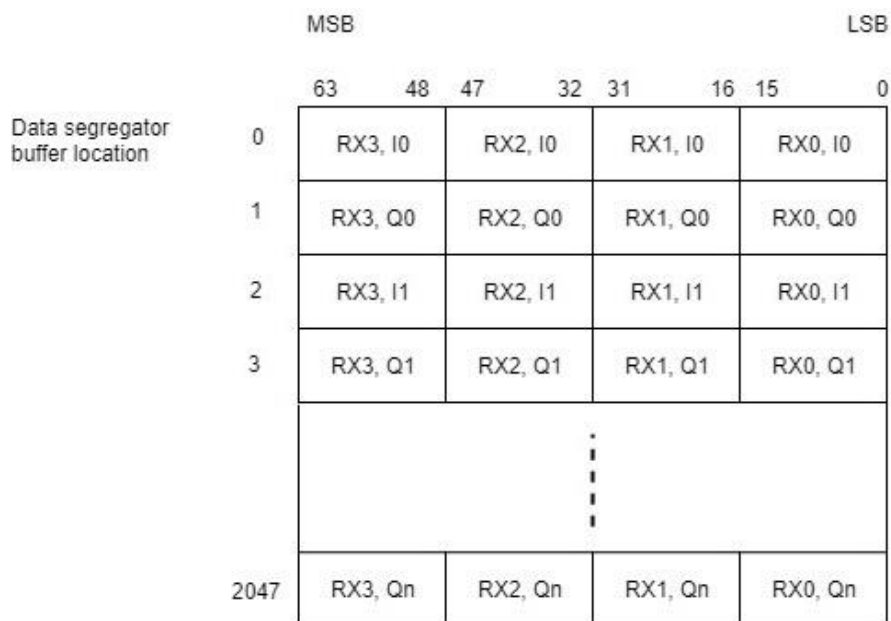


**Figure 31 Ethernet payload format – [Multi mode 2 -lane]**

- Below two figures shows the data format which is stored in data segregator buffer. This data will be read as 8-bit (starting from LSB to MSB) and transmitted over Ethernet.


**Figure 32 2-lane data format**

1.


**Figure 33 4-lane data format**

#### **Raw Mode:**

- Data segregator module reads data from internal buffer and sends following information to UDP packet creation module.
  - UDP port number – it is always ADC data port number for raw mode
  - 4 bytes of sequence number for every packet
  - 6 bytes of byte count – provides information about number of bytes transmitted till last packet.

- Payload data – maximum 1456 bytes
- In raw mode, UDP packet length is maintained to 1456 bytes.
- To ensure UDP packet length of 1456 bytes, this module waits for 1456 bytes of data to be available in buffer and then starts reading data from buffer.
- If waiting time is more than 2s and data available in buffer is less than 1456 bytes, this module reads the data available in buffer and sends to Ethernet interface module.

#### **Multi-Mode:**

- Data segregator module reads data from internal buffer and checks for header types mentioned in Table-27.
- Table-27 shows different data types which are supported by RF Data Capture card.

Identifier (64 bits)	UDP port
0x0ADC0CDA0ADC0CDA	"Raw data port"
0x0ADD0DDA0ADD0DDA	
0x0ADE0EDA0ADE0EDA	
0x0ADF0FDA0ADF0FDA	
0x0CC909CC0CC909CC	"Raw data port" + 1
0x0CC808CC0CC808CC	"Raw data port" + 2
0x084f0f48084f0f48	"Raw data port" + 3
0x0C67076C0C67076C	"Raw data port" + 4

**Table 27 Data types supported in DCA1000EVM**

- Table-28 and Table-29 shows multi-mode ADC data format for 4-lane and 2-lane boards.
- In multi-mode, data is received along with header which contains data identifier and length field.
- Length field provides the byte count which will be received followed by the header.
- After processing, both header and data are transmitted to GUI.

Lane No.	Clock 1	Clock 2	Clock3 ..... Clock N
Lane 0	0x0CDA	0x0ADC	ADC DATA
Lane 1	0x0CDA	0x0ADC	
Lane 2	Len [11:0]	Len [23:12]	
Lane 3	Reserved [47:24]	Reserved [47:36]	

**Table 28 Multi-mode ADC data format for 4-lane**



Lane No.	Clock 1	Clock 2	Clock 3	Clock 4	Clock5 ..... Clock N
Lane 0	0x0CDA	0x0ADC	Len [11:0]	Len [23:12]	ADC DATA
Lane 1	0x0CDA	0x0ADC	Reserved [35:24]	Reserved [47:36]	

**Table 29 Multi-mode ADC data format for 2-lane**

- Data segregator module sends following information to UDP packet creation module.
  - UDP port number – it can be ADC/CP/CQ/R4F/DSP port number, depends on the header type received.
  - 4 bytes of sequence number – provides packet number. Separate sequencer count is available for each data port.
  - 6 bytes of byte count – provides information about number of bytes transmitted till last packet. Separate byte count is available for each data port.
  - Payload data – maximum 1456 bytes.
- Minimum byte count for processing the data is set to 48 bytes since the data separated mode has been tested with minimum capture size of 48 bytes.
- This module waits for minimum 48 bytes of data to be available in buffer and then starts reading data from buffer.
- If waiting time is more than 2s and data available in buffer is less than 48 bytes, this module reads the available data and processes it (data type and length field identification).

UDP packet length is decided based on the length field available in multi-mode header. If length of the frame is more than 1456, the frame payload data is split into multiple packets. The next frame will be sent as a new packet with the multi-mode header ID.

### **3 Software Reference**

Data structures, APIs, Command and error codes details are captured in the doxygen provided with the release.