SPECIFICATION FOR EXTERNAL USE of I/O DLLs TO COMMUNICATE TO FDM DEVICE

This document explains the interface between a generic Software Defined Radio and Windows DLLs implemented for communication to FDM-S1/FDM-S2 device. The DLL is the layer-code between software and hardware, it connects to FDM-S1/FDM-S2 device and implements the real-time data acquisition.

All the functions exported from FDM-DLLs have the same calling convention described in ***"Winrad-Specification for external I/O DLL"*** document. Then for each function exported, input arguments and return values are maintained.

Please read ***"Winrad-Specification for external I/O DLL"*** document to import DLL functions correctly on top-level software and to execute a dynamic-link to DLL.

The "Modified Synopsys" paragraph describes the functions that differ from the specification for generic external I/O DLL exposed on ***"Winrad- Specification for external I/O DLL"*** document: for example in `SetCallback(…)` function pointer, the argument `status` is coded in a different way in FDM-DLL.
Please read "Modified Synopsys" paragraph to check the differences.

A single DLL can interface only one FDM device to top level software and acquire data at fixed output data rate.
Then to communicate to FDM-S1 or FDM-S2, two different DLLs must be implemented: one for FDM-S1 interface and connection and the other for FDM-S2 interface and connection.
The same must be done to have acquisition at more the one output data rate per FDM-device. For FDM-S2, there are listed the last 32-bit DLL versions implemented at output data rate specification:

- `ExtIO_ELAD_FDMS2_192k_v3_04.dll` for 192 kSample output data rate.
- `ExtIO_ELAD_FDMS2_384k_v3_04.dll` for 384 kSample output data rate.
- `ExtIO_ELAD_FDMS2_768k_v3_04.dll` for 768 kSample output data rate.
- `ExtIO_ELAD_FDMS2_1536k_v3_04.dll` for 1536 kSample output data rate.
- `ExtIO_ELAD_FDMS2_3072k_v3_04.dll` for 3072 kSample output data rate.
- `ExtIO_ELAD_FDMS2_6144k_v3_04.dll` for 6144 kSample output data rate.

The "DLL interface to top-level software and FDM connection" paragraph explains the correct calling sequence to attach and connect on FDM-device.

The "DLL dialog box" paragraph describes the dialog box that appears when DLL is loaded from top-level software. This dialog box is used to configure FDM hardware.

Modified Synopsys

```
bool InitHW(char *name, char *model, int& type)
```

This entry is first called by top-level software at startup time and returns some DLL parameters as device name, device model, and data format. For FDM-S1/FDM-S2, this call returns following values:

    name = "FDM-S1"/"FDM-S2"
    model = "1"
    type = 7

`type` argument means data are in 32-bit float format, little endian.
If the function returns `true` the initialization is done, otherwise the initialization fails.

```
bool OpenHW(void)
```

This entry is called to connect to FDM-S1/FDM-S2 device to USB. Then this function must be invoked each time the user specifies that top-level software should receive its audio data input through the FDM-S1/FDM-S2 hardware.
If it returns `true`, FDM-S1/FDM-S2 is connected and the device is ready for data acquisition.
If there is more than one FDM-device connected to PC, this function connects the first FDM device found (multi device connection is not supported with these DLL specification).

This entry must be invoked after `InitHW(…)` function.

```
int StartHW(long LOfreq)
```

This function must be call each time user wants to start acquisition (for example pressing a start button on a GUI or windows form developed at top-level software).
The function argument is:

- `LOfreq` specifies the local oscillator frequency in FDM-S1/FDM-S2 (in Hz).

The function set local oscillator frequency in FDM-S1/FDM-S2 when ev
Return value means how many complex samples are returned each time the callback function is invoked. For FDM-S1/FDM-S2 return value changes based on DLL output data rate:

- For 192k DLL return value is 12288 complex samples.
- For 384k DLL return value is 12288 complex samples.
- For 768k DLL return value is 12288 complex samples.
- For 1536k DLL return value is 12288 complex samples.
- For 3072k DLL return value is 49152 complex samples.
- For 6144k DLL return value is 49152 complex samples.

```
void StopHW(void)
```

This function stops data acquisition and it could be invoked pressing a stop button from a GUI or windows form. There aren't arguments or return values.

```
void CloseHW(void)
```

This function close FDM-S1/ FDM-S2 hardware and free memory from variables allocated. Usually it follows `StopHW()`.There aren't arguments or return values.

```
int SetHWLO(long freq)
```

This function is used to set FDM-S1/FDM-S2 local oscillator frequency expressed in unit of Hertz. This entry point could be called at each change of the local oscillator on a GUI or windows form at top-level software.
The function argument is:

- `freq` specifies the local oscillator frequency for FDM-S1/ FDM-S2 (Hz).

The function return value is 0 or 1: if return value is 0, the function is done and local oscillator frequency is set; otherwise function fails and local oscillator frequency is not set.

```
long GetHWLO(void)
```

The function return value is the current local oscillator frequency on FDM-S1/ FDM-S2 expressed in unit of Hertz.

```
long GetHWSR(void)
```

This function is used to know DLL output sample rate value (in Hz). Sample rate is a fixed parameter for DLL of FDM-S1/FDM-S2.
Then, in order to have different output sample rate values, different DLL must be loaded by top level software:

- In `ExtIO_ELAD_FDMS*_192k_v*` DLL, return value is 192000 Hz.
- In `ExtIO_ELAD_FDMS*_384k_v*` DLL, return value is 384000 Hz.
- In `ExtIO_ELAD_FDMS*_768k_v*` DLL, return value is 768000 Hz.
- In `ExtIO_ELAD_FDMS*_1536k_v*` DLL, return value is 1536000 Hz.
- In `ExtIO_ELAD_FDMS*_3072k_v*` DLL, return value is 3072000 Hz.
- In `ExtIO_ELAD_FDMS*_6144k_v*` DLL, return value is 6144000 Hz.

```
long GetTune(void)
```

Entry point not implemented.

```
void GetFilters(int& loCut, int& hiCut, int& pitch)
```

Entry point not implemented.

```
char GetMode(void)
```

Entry point not implemented.

```
void ModeChanged(char mode)
```

Entry point is implemented but not used.

```
int  GetStatus(void)
```

Entry point is implemented but not used and returns 0.

```
void  ShowGUI (void)
```

Entry point is implemented but not used.

```
void  HideGUI (void)
```

Entry point is implemented but not used.

```
void  IFLimitsChanged (long low, long high)
```

Entry point is implemented but not used.

```
void  TuneChanged (long freq)
```

Entry point is implemented but not used.

```
void  RawDataReady (long samprate, int *Ldata, int *Rdata, int numsamples)
```

Entry point is implemented but not used.

```
void SetCallback (void (*Callback)(int, int, float,short *))
```

This entry is a function pointer: top-level software uses this call to pass the address of a owner function (for example defined as `extIOCallback(…)`) to DLL. By DLL side, `SetCallback(…)` attaches this function pointer to a routine that fills arguments with data received from FDM-S1/ FDM-S2: when a new data buffer is acquired, the DLL invokes this routine and the result is to pass data to top-level software.

Software code for `Callback` function that DLL expects to invokes should be defined as:

```
float DataSwInBuffer[BUFFER_SIZE*2];
//NB: BUFFER_SIZE means the number of complex samples (or IQ pairs).
//then BUFFER_SIZE must be equal to StartHW(…) return value

void extIOCallback (int cnt, int status, float IQoffs, short *IQdata))
{
        memcpy(DataSwInBuffer, IQdata, BUFFER_SIZE*sizeof(float)*2);

        //to do: data processing on acquired DataSwInBuffer …

}
```

Function arguments are defined as follow:

- `cnt` is the number of short format samples returned. Data acquired from FDM-S1/ FDM-S2 are in 32-bit-float I and 32-bit-float Q format or 64-bit-float for complex sample, so the total number of samples (in short format) are BUFFER_SIZE*4; if data are acquired in 12288 sample-sized buffers, BUFFER_SIZE must be 12288 and then `cnt` is equal to 12288*4.
- `status` is a 32 bit integer variable and it is used to notify hardware status and usb-communication status to software.
  - `status<7:0>` bits (or least significant byte) mean something is failed in usb-communication. For usb-communication error decode refer to “USB Error Decode” paragraph.
  - `status<8>` bit means ADC overrange on FDM-S1/FDM-S2 device.
  - `status<9>` bit means if mute is enabled on FDM-S1/ FDM-S2. Mute is enabled (status<9>=1) when an active-low signal is present on pin n° 6 of FDM-S1/ FDM-S2 external connector (EXT I/O connector on device). Mute is disabled by default.
- `IQoffs` is an offset value to minimize DC offset on hardware; this value is unused and sets to 0.0.

- `*IQdata` is data buffer pointer where DLL expects to place samples acquired from FDM-S1/FDM-S2 in an interleaved format (I-Q-I-Q). The `*IQdata` pointer must point to an array sized with `StartHW(…)` return value number of complex samples.

DLL interface to top-level software and FDM connection

To correctly connect FDM-S1/ FDM-S2 and acquire data, top-level software must invoke functions in this order:

```
InitHW()
SetCallback(…)
OpenHW()
StartHW(…)
StopHW()
CloseHW()
```

Changes on FDM-S1/ FDM-S2 hardware must be done after `OpenHW()` call and before `CloseHW()` call: this condition means that FDM device is opened and connected to USB. The routines must satisfied this condition are `StartHW(…)`,`StopHW()`, `SetHWLO(…)`, `GetHWLO(…)`, because these functions act directly on FDM hardware.

DLL dialog box

When DLL is loaded by top-level software, a dialog box appears as depicted in Figure 1.

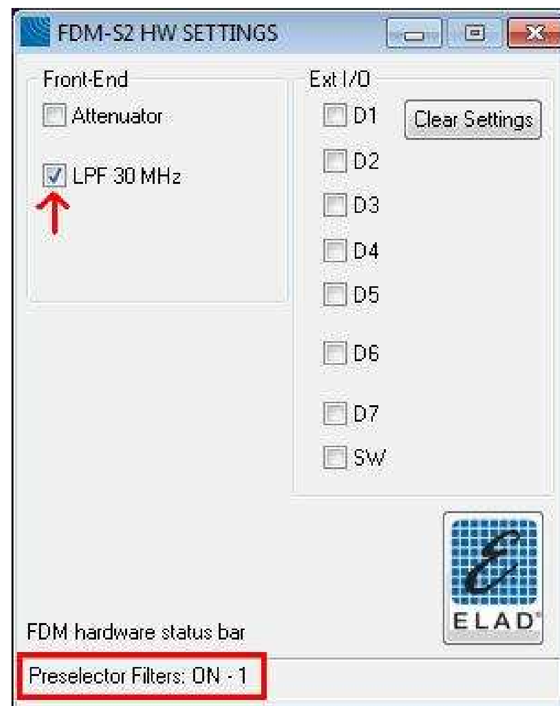

Figure 1: pre-selection filters on.

If the device connected is FDM-S2, the “LPF 30 MHz” check box is checked by default. This option checked means that the pre-selection filters on FDM-S2 are enabled and user can receive correctly signals on HF or VHF channels (see red signs on Figure 1).

If “LPF 30 MHz” check box is not checked, pre-selection filters on FDM-S2 are bypassed (see red signs on Figure 2).

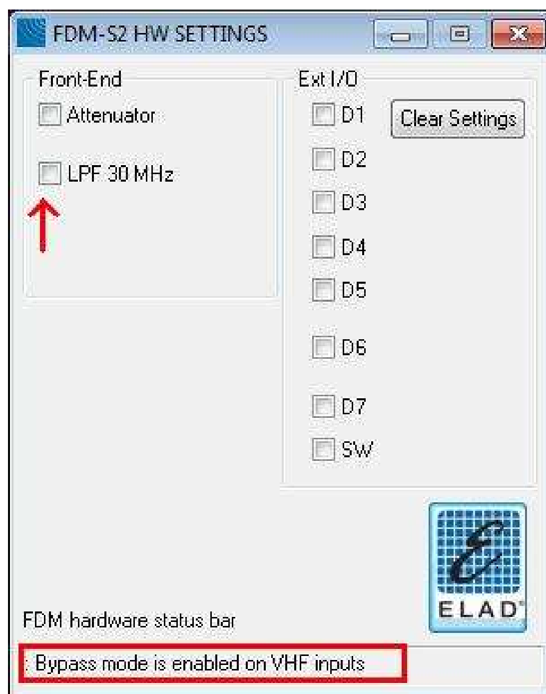

Figure 2: Bypass pre-selection filters

In the same way, user can set/unset FDM-S2 attenuator or set/unset lines on EXT I/O connector.

USB Error Decode

```
//reserved for future changes//


#define STATUS_OK                                         0
//fdm-device is connected to USB socket, fpga is loaded and configured, adc is correctly configured:
//fdm is ready to start acquisition.

#define ERROR_FULL_STATUS_FROM_USB                        1
#define ERROR_EMPTY_STATUS_FROM_USB                       2
#define ERROR_READ_STATUS_FROM_USB                        3
#define ERROR_INIT_XILINX                                 4
#define ERROR_PROG_XILINX_INCOMPLETE                      5
#define ERROR_PROG_XILINX_DONE_FALSE                      6
#define ERROR_INIT_CY_FIFO                                7
#define ERROR_START_CY_FIFO                               8
#define ERROR_STOP_CY_FIFO                                9
#define ERROR_RESET_CY_FIFO                               10
#define ERROR_TUNE_ON_XILINX_FPGA                         11
#define ERROR_INIT_EXT_FILTERS                            12
#define ERROR_MEM_ALLOCATION                              13
#define ERROR_EE_RD_WR                                    14
#define ERROR_MANAGE_XILINX_FPGA                          15
```

```c
#define ERROR_INIT_USB_DEVICE                                    16
#define ERROR_INIT_FILTERS_TABLE                                 17

#define ERROR_CREATE_THREAD                                      20
#define ERROR_SET_EXT_FILTERS                                    21
#define ERROR_WRITE_ADC_REG                                      22
#define ERROR_READ_ADC_REG                                       23

#define ERROR_READ_EE_LEVEL_OFS                                  24
#define ERROR_WRITE_EE_LEVEL_OFS                                 25
#define ERROR_READ_EE_LOFREQ_OFS                                 26
#define ERROR_WRITE_EE_LOFREQ_OFS                                27
#define ERROR_READ_EE_FSFREQ_OFS                                 28
#define ERROR_WRITE_EE_FSFREQ_OFS                                29
#define ERROR_READ_EE_HW_VER                                     30
#define ERROR_WRITE_EE_HW_VER                                    31
#define ERROR_READ_EE_PDC                                        32
#define ERROR_WRITE_EE_PDC                                       33
#define ERROR_READ_EE_SN                                         34
#define ERROR_WRITE_EE_SN                                        35
#define ERROR_GET_FW_USB_VER                                     36

#define ERROR_IS_NOT_OPEN_HW                                     40
#define ERROR_IS_START_COMM_ON                                   41
#define ERROR_HW_CFG_OUT_OF_RANGE                                42
#define ERROR_UNCODED_LEVEL_OFS                                  43
#define ERROR_DAC_AUDIO_EXIST                                    44
#define ERROR_DAC_AUDIO_ENABLE                                   45
#define ERROR_SET_HW_ATT                                         46
#define ERROR_SET_HW_LP                                          47
#define ERROR_SET_HW_PRE                                         48

#define ERROR_OPEN_CFG_TXT_FILE                                  50
```