MultiHarp 160

IP and Software Library for
FPGA-based time tag processing

EFI - External FPGA Interface for MultiHarp 160

Programming Reference Handbook

Document version 1.0.3

Table of Contents

1. Introduction

The MultiHarp 160 is a cutting edge TCSPC system with USB interface. Its new integrated design provides a flexible number of input channels at reasonable cost and allows innovative measurement approaches. The timing circuits allow high measurement rates up to 78 million counts per second (Mcps) with an excellent time resolution and a record breaking deadtime of 650 ps. The modern USB 3.0 interface provides very high throughput as well as ‘plug and play’ installation. The input triggers are adjustable for a wide range of input signals providing programmable level triggers for both negative and positive going signals. These specifications qualify the MultiHarp 160 for use with most common single photon detectors such as Single Photon Avalanche Diodes (SPADs), Superconducting Nanowire Single Photon Detetors (SNSPD), and Photomultiplier Tube (PMT) modules (via preamplifier). Depending on detector and excitation source the width of the overall Instrument Response Function (IRF) can be as small as 80 ps FWHM. The MultiHarp 160 can be purchased with 16, 32, 48 or 64 timing inputs and one synchronization (sync) input. The use of these inputs is very flexible. In fluorescence lifetime applications the sync channel is typically used as a synchronization input from a laser. The other inputs are then used for photon detectors. In coincidence correlation applications all inputs including the sync input can be used for photon detectors.

The MultiHarp 160 USB 3 interface, together with its TTTR measurement modes, enables a great performance and flexibility trade-off for most use cases. However, when targeting very high counts per second or large experimentation setups with up to 64 channels, it may not provide enough bandwidth. In some setups it may also be undesirable to only process data in the standerd TTTR file formats. Finally, the computer attached to the USB interface must keep up to the data generated by the MultiHarp 160, which can be a challenge when running complex real-time data analysis algorithms.

The External FPGA Interface (EFI) for MultiHarp 160 addresses those challenges. It uses one or more high speed serial link cables to transfer data to an external FPGA. On such a board custom processing and I/O can be performed with virtually unlimited flexibility. This solves the above mentioned problems, as (1) high speed serial links increase the bandwidth by an order of magnitude, (2) event data can be processed in a raw format and (3) complex processing algorithms can be performed in real-time.

In order to streamline the development of EFI solutions, PicoQuant provides an initial set of gateware and software IP. Custom logic, which can be written in VHDL, Verilog or any other Xilinx Vivado supported language, only needs to be connected to a set of high-level data stream interfaces. The user can choose to either receive the raw timing data or the pre-processed TTTR T2 or T3 mode time tags. Through a loopback interface, custom data generated by the FPGA can be transmitted back trough the USB3 interface. The FPGA logic can be easily verified against PTU files using the supplied simulation library. Deep customization of the gateware IP is supported, as all required sources for the external FPGA are freely available.

The first part of this programming reference describes how to get started with the demo design. It is recommended to read this part in its entirety before starting development with the External FPGA Interface. The second part of this document contains a reference of the interface semantics of the gateware, how to use the simulation library and a documentation of the External FPGA Interface specific functions of the MultiHarp software programming library MHLib.

2. Getting started with the External FPGA Interface

Compared to software analysis of time series data, FPGA based processing has many advantages. However, every FPGA based system requires thorough understanding and careful consideration of the design parameters in order to make good use of it. This can be a strain on developer resources. In order to jump-start the development with the EFI, this chapter is in the format of a tutorial that familiarizes the user with the capabilities of the system.

In this tutorial we first show how to set up the MultiHarp 160 with the Digilent Genesys 2 FPGA development board. Furthermore, a Vivado example project shows how to access the TTTR data streams in the FPGA. The tutorial thus also serves as a good starting point for developing custom EFI-based solutions. We therefore strongly recommend starting with this tutorial when you first use the External FPGA Interface.

2.1. Performance numbers of the example design

The following table lists the maximum event rates and the latency depending on the configuration of the MultiHarp 160. The latency is measured by subtracting the arrival time of the event pulse at the MultiHarp 160 from the arrival time of the corresponding TTTR tag in the external FPGA.

MultIHarp 160 Configuration	Maximum Event Rate	Latency
Histogramming Mode	Not supported	Not supported
T2 Mode	200,000,000 Events/second	4.5 us to 5.0 us
T3 Mode	200,000,000 Events/second	4.5 us to 5.5 us
T2 Direct Mode	78,000,000 Events/second for the Sync + 200,000,000 Events/second shared among each horizontal row of 8 inputs	Sync: 1.7 us to 1.8 us Others: 0.8us to 1.2 us

2.2. System Requirements of the example design

In order to follow this guide the following requirements should be met:

- The External FPGA Interface 1.0.3 source package.
- A Windows 10 x64 computer with the MultiHarp 3.0 driver installed.
- Vivado 2018.3 with an activated license for the XC7K325T FPGA. Newer versions of Vivado may also work, but are not recommended.
- A C/C++ compiler that supports at least C++14, such as gcc, clang or MSVC..
- A Digilent Genesys 2 FPGA development board.
- The Genesys 2 EFI FMC Adapter and MultiHarp 160 EFI cable(s).
- A signal source for the MultiHarp 160 for testing purposes (e.g. PicoQuant PDL 800-D).

This guide further assumes that the programming library MHLib and PicoQuant 160 documentation has been understood. The documentation of the Digilent Genesys 2 FPGA development board reference manual can be found under the following link: https://reference.digilentinc.com/reference/programmable-logic/genesys-2/reference-manual.

2.3. Setting up the Hardware

1. Turn on the MH160 and connect the USB3 cable to the computer.
2. Plug the Genesys 2 EFI FMC Adapter into the Genesys 2 board.
3. Connect the Genesys 2 EFI FMC Adapter and the MultiHarp 160 ext link port on the back of the device using the MultiHarp 160 EFI cable.
4. Connect the USB JTAG and USB UART cables from the Genesys 2 board to the PC and turn on the board.

2.4. Generating the Gateware

1. Ensure that the license file is installed correctly by searching for the XC7K325T FPGA in the Xilinx license manger.
2. Install the board files for the Genesys 2 board. See the Digilent documentation for further details.
3. Extract the External FPGA Interface source package to a destination of your choice.
4. Extract the gateware sources in the `<package location>/gateware/` folder.
5. Open the Vivado project located in the folder `<package location>/gateware/ext_genesys2_r01`. It is recommended to not nest Vivado projects to deep into the filesystem, as Windows may be constrained by path length limitations.
6. Generate the bitstream.
7. Open the device manager, select auto connect and program the FPGA.

2.5. Testing the System

1. Ensure that the MultiHarp driver version 3.0 is installed correctly and that the system can connect to the device by opening the MultiHarp application.
2. Connect your signal sources to the MultiHarp.
3. Go to `<package location>\software\API_example` and compile the C code using MHLib. Detailed instructions are provided in the MHLib programming guide. The code may already be familiar to you, as it is based on the MHLib example code.
4. Run the generated exe file.
5. Check if the software raises any errors and if the reported count rates are correct.
6. Upon pressing a key in the console window the software will start a measurement for 50 milliseconds. The external FPGA mode and loopback mode are hardcoded in the API_example. The generated data is sent to external FPGA and count rates per channel are calculated. Those are periodically sent back to the host using the loopback interface, if it is enabled. The returned data is displayed in the console window. If T2 or T3 loopback modes are used, the T2 or T3 records are returned instead.
7. Make sure that all combinations of channels, external FPGA mode and loopback mode that you require are working and returning sensible count rates.
8. This concludes the setup and getting started guide for the External FPGA Interface.

2.6. Modifying the Example Design

The example design is a great starting point for building your own data processing solution with the EFI. It is therefore recommended to familiarize yourself with VHDL code in the Vivado project and the C code for the host application.

The relevant source files for the example design are `usr_application_example.vhd` and `ext_genesys2_r01_top.vhd`. The usr_application_example file contains all the logic interfacing the EFI. The `ext_genesys2_r01_top` connects the user example and the EFI IP. It also implements a fan speed limiter in order to reduce the noise level of the system. When extending and adding VHDL files you should make sure to include the `pq_extfpga_lib` library and `pq_extfpga_pkg` package imports for all contexts.

The EFI uses an embedded Microblaze controller for various tasks. It is recommended to use the ELF file that is supplied with the example project, as the inner workings of the EFI are not guaranteed to remain identical between revisions of this platform. However, should the need to recompile or modify the ELF file arise, you can find the sources used to build the file in the folder `<package location>/gateware/sdk`.

2.7. Simulating the Design

The EFI design package contains a VHDL library for large scale simulation of TTTR processing logic. The code can be found in `<package location>/gateware/hdl_src/pq_mhsim_lib/`. This library contains a simulation model of the MultiHarp 160, that can be connected directly to the top level entity of your FPGA design. Photon and marker events can be fed into the simulation model using ptusim files. Using the CPP file located in `pq_mhsim_lib/sim/ptusim` it is possible to convert PTU files into PTUSIM files.

2.8. Porting the design to other FPGAs

The EFI design package can be easily ported to other FPGA development boards and systems, as all required source are included and can be modified freely. It will require some degree of work on the internals of the package, depending on how similar the desired FPGA is to the Genesys 2 board. The following lists the potentially required changes depending on what kind of FPGA you want to use.

FPGA boards that contain the XC7K325T FPGA:

- The XDC file must be changed to reflect the different pinout.
- A new EFI-FMC connector may be required. You can fabricate your own using the schematics included in the hardware folder, or you can contact PicoQuant support for assistance.
- Note that at least 2 GTX transceiver are required for the base functionality. If you wish to use the T2DM channels, then one GTX transceiver is required per channel.

FPGA boards that contain a different Xilinx FPGA:

- You must ensure that the “Aurora 8b/10b” IP-Core is supported for your FPGA. If the transceivers of the FPGA are not “7 Series-GTX”, you must change the portion of the EFI-Design that connects the transceivers with the Aurora-IP. You must regenerate the Aurora-IP.
- The XDC file must be changed to reflect the different pinout.
- A new EFI-FMC connector may be required. You can fabricate your own using the schematics included in the hardware folder, or you can contact PicoQuant support for assistance.
- Note that at least 2 GTX transceiver are required for the base functionality. If you wish to use the T2DM channels, then one transceiver is required per channel.

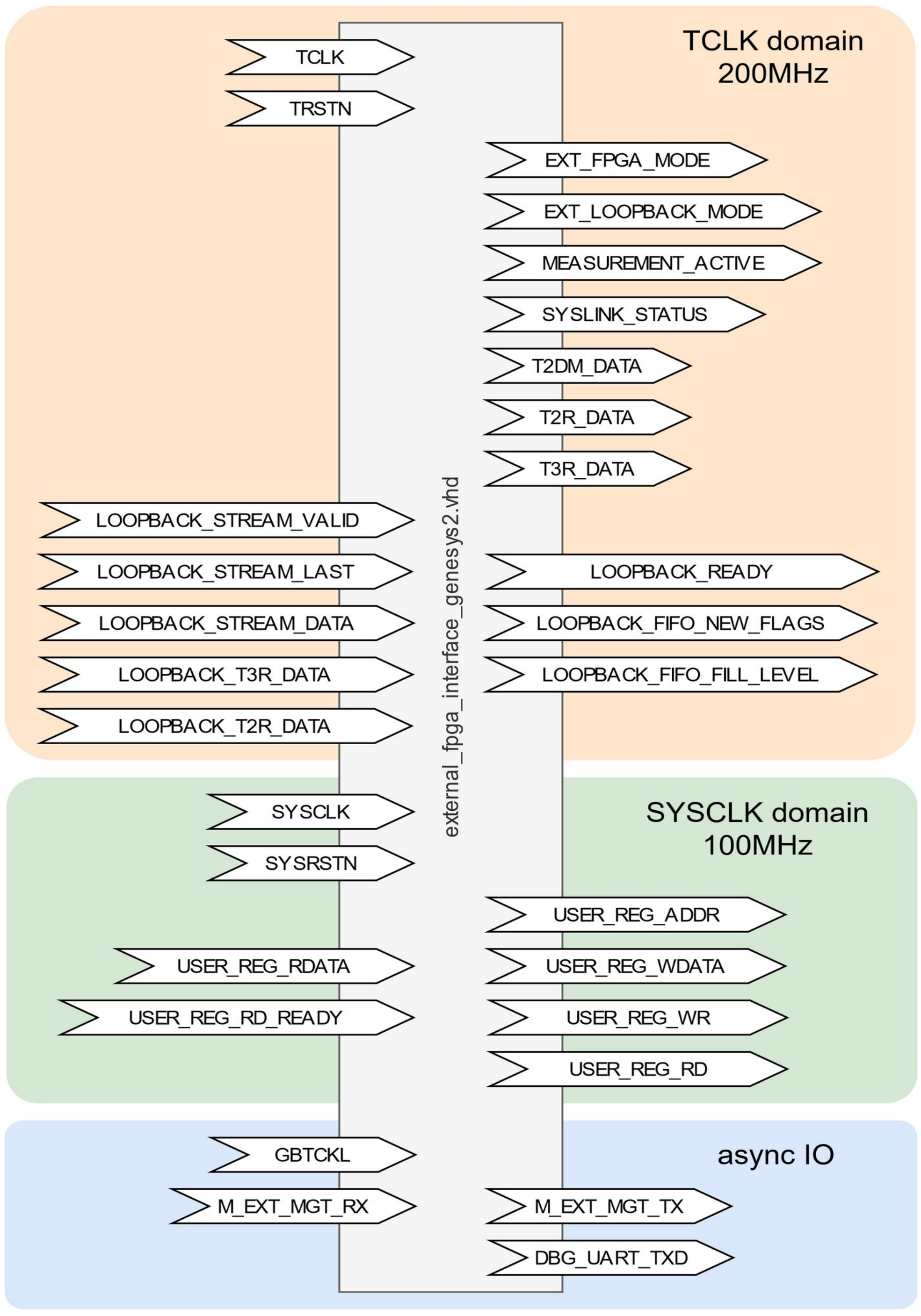
FPGA from different vendors or other semiconductors:

- Please ensure that the device supports the PHY protocol outlined in the Aurora 8b/10b standard.
- You can use the provided VHDL-Code and simulation environment as reference to implement your endpoint for the External FPGA Interface.

3. External FPGA Interface IP Reference

The External FPGA Interface manages all communication with the MultiHarp 160 transparently. The user can focus on implementing the data processing and I/O functions that are specific to her use case. The primary interfaces of the EFI are the high speed tag streams for T2, T3 and T2DM channels, the USB3 loopback interface and the user register configuration interface. The IP is provided as VHDL source.

The following image and table list all the signals and their properties for the EFI IP. The behavior of the signals is defined in the subsequent subchapters.



Signal Name	Signal Type	Direction	Domain	Description
TCLK	std_logic	IN	TCLK	200 MHz clock
TRSTN	std_logic	IN	TCLK	Active-low reset
EXT_FPGA_MODE	std_logic_vector(1:0)	OUT	TCLK	Defined in chapter 3.1
EXT_LOOPBACK_MODE	std_logic_vector(1:0)	OUT	TCLK	Defined in chapter 3.1
MEASUREMENT_ACTIVE	std_logic	OUT	TCLK	Defined in chapter 3.1
T2DM_DATA	t2dm_rec_vec(0:8)	OUT	TCLK	Defined in chapter 3.2
T2R_DATA	t2r_rec	OUT	TCLK	Defined in chapter 3.3
T3R_DATA	t3r_rec	OUT	TCLK	Defined in chapter 3.3
LOOPBACK_STREAM_DATA	std_logic_vector(31:0)	IN	TCLK	Defined in chapter 3.4
LOOPBACK_STREAM_VALID	std_logic	IN	TCLK	Defined in chapter 3.4
LOOPBACK_STREAM_LAST	std_logic	IN	TCLK	Defined in chapter 3.4
LOOPBACK_READY	std_logic	OUT	TCLK	Defined in chapter 3.4
LOOPBACK_T2R_DATA	t2r_rec	OUT	TCLK	Defined in chapter 3.4
LOOPBACK_T3R_DATA	t3r_rec	OUT	TCLK	Defined in chapter 3.4
LOOPBACK_FIFO_NEW_FLAGS	std_logic	OUT	TCLK	Defined in chapter 3.5
LOOPBACK_FIFO_FILL_LEVEL	std_logic_vector(15:0)	OUT	TCLK	Defined in chapter 3.5
SYSLINK_STATUS	system_link_status_rec	OUT	TCLK	Defined in chapter 3.6
SYSCLK	std_logic	IN	SYSCLK	100 MHz clock
SYSRSTN	std_logic	IN	SYSCLK	Active-low reset
USER_REG_ADDR	std_logic_vector(31:0)	OUT	SYSCLK	Defined in chapter 3.7
USER_REG_WDATA	std_logic_vector(31:0)	OUT	SYSCLK	Defined in chapter 3.7
USER_REG_RDATA	std_logic_vector(31:0)	IN	SYSCLK	Defined in chapter 3.7
USER_REG_WR	std_logic	OUT	SYSCLK	Defined in chapter 3.7
USER_REG_RD	std_logic	OUT	SYSCLK	Defined in chapter 3.7
USER_REG_RD_READY	std_logic	IN	SYSCLK	Defined in chapter 3.7
GBTCLK[0/1]_[P/N]	std_logic	IN	Async IO	Defined in chapter 3.8
M_EXT_MGT_TX[P/N]	std_logic_vector(1:0)	OUT	Async IO	Defined in chapter 3.8
M_EXT_MGT_RX[P/N]	std_logic_vector(9:0)	IN	Async IO	Defined in chapter 3.8
DBG_UART_TXD	std_logic	OUT	Async IO	Defined in chapter 3.8

3.1. TCLK Domain Status Signals

The TCLK domain contains three status output signals. The external FPGA can process data coming from the MultiHarp 160 in a variety of formats. The T3 and T2 data formats represent the data in an identical way to how they would be received in the MHLib DLL and how they would appear in a PTU file. The T2DM, short for T2 Direct Mode format, uses a variation of the T2 format. Using the EXT_FPGA_MODE signal, the external FPGA logic can adapt to different timetag sources. It is encoded as follows:

EXT_FPGA_MODE(1:0)	Name	Description
00	efi_mode_off	The external FPGA is not used.
01	efi_mode_t2raw	The external FPGA receives data In the T2DM format.
10	efi_mode_t2r	The external FPGA receives data In the T2 DLL format.
11	efi_mode_t3r	The external FPGA receives data In the T3 DLL format.

The external FPGA can send data from to the PC using the USB 3.0 interface of the MultiHarp. This is referred to as the loopback stream. Different data sources can be set up for this stream. The EXT_LOOPBACK_MODE signal is encoded as follows:

EXT_LOOPBACK_MODE(1:0)	Name	Description
00	efi_loopback_off	The loopback interface is turned off.
01	efi_loopback_user	The loopback interface transports the LOOPBACK_DATA data.
10	efi_loopback_t2	The loopback interface transports LOOPBACK_T2R data.
11	efi_loopback_t3	The loopback interface transports LOOPBACK_T3R data.

The signals EXT_FPGA_MODE and EXT_LOOPBACK_MODE are always valid and reflect the parameters supplied to the `MH_ExtFPGASetMode()` MHLib call.
The MEASUREMENT_ACTIVE signal informs the external FPGA function whether a measurement is going on or not. It is always valid.

3.2. T2DM Stream

The T2DM stream data is a key feature of the External FPGA Interface. It uses many serial high speed links to connect the external FPGA to the time measurement logic as directly as possible. Using T2DM, it is possible to process more than 1 Gigaevent per second using the MultiHarp 160 with 64 channels. It also reduces the latency between measurement and availability in the FPGA by 80% compared to the T2/T3 streams.

The streams of the T2DM-Mode are based on the semantics of the T2-Mode. Please refer to the MultiHarp User's Manual for details on this mode. The following only explains the encoding differences to the T2-Mode.

In T2DM mode the user logic interfaces with 3 to 9 streams. The stream zero only carries the events from the sync input, as well as the marker inputs. All other streams carry the events of 8 inputs each, with stream one containing events from the inputs 1 to 8, stream two containing events from the inputs 9 to 16, and so on.

The T2DM events are encoded in the `t2dm_rec` type. There are four different groups of code points in this encoding, which are represented by the `t2dm_tag_type`. Using the `get_t2dm_tag_type()` call the tag type of a `t2dm_rec` can be easily queried. The following table lists the different possible encodings and their meanings.

wr	ovfl	channel(5:0)	get_t2dm_tag_type	Description
0	any	any	OTHER_TAG	Invalid
1	0	00_00_00	EVENT_TAG	Event on local Channel 1. Never used on stream 0.
		00_00_01		Event on local Channel 2. Never used on stream 0.
		0X_XX_XX		Event on local Channels 3-31 correspondingly. Never used on stream 0.
		01_11_11		Event on local Channel 32. Never used on stream 0.
		10_00_00		Event on the Sync Channel. Only used on stream 0.
		10_AB_CD	MARKER_TAG	Marker 4 event when A = 1, Marker 3 event when B = 1 Marker 2 event when C = 1, Marker 1 event when D = 1. Only used on stream 0.
		11_00_00	OTHER_TAG	Start of measurement for all channels on this stream.
		11_00_01		End of measurement for all channels on this stream.
	1	00_00_00		Unused
		...		
		11_11_11	OVERFLOW_TAG	One overflow for all channels on this stream.

EVENT_TAGs represent photon events on the MultiHarp inputs. The tag value shall be interpreted as the tag value in the T2 format. The encoding of the channel uses a concept called local channel. For the Multi-Harp160, given the stream number S and the local channel L the input number I can be computed by I=(S-1)*8+L.

MARKER_TAGs represent events on the Marker inputs 1 to 4. If two Marker events are triggered at the same time, they can be encoded in a single `t2dm_rec`.

OTHER_TAGs represent unused code points and miscellaneous status information, such as the start and end points of measurements for all channels of the corresponding stream. The start and end tags can be used to find the timetag start value at the beginning or end of a measurement.

OVERFLOW_TAGs represent an overflow similarly to how the T2 uses them. The overflow is only valid for the channels of the corresponding stream. Only one overflow is encoded by a OVERFLOW_TAG.

An array of `t2dm_rec` types is denoted by the type `t2dm_rec_vec` and may have arbitrary dimensions. The following table lists the fields, their meaning and their corresponding bit widths of the `t2dm_rec` type.

Field name	Bit width	Description
wr	1	Indicates whether record is valid
ovfl	1	Indicates special record values
channel	6	Indicates channel and special record values
tag	17	Encodes the time tag of the event

3.3. T2/T3 Stream

The T2 and T3 modes transport data streams in a format that is described in the MultiHarp manual. Another good starting point for understanding these formats are the PTU file demos installed together with the regular MultiHarp software. The event records are encoded using the `t2r_rec` and `t3r_rec` correspondingly. A record is only valid when its `wr` value is set to ‘1’.

The streams may present valid data that does not fit any encoding outlined in the PTU file documentation. In the interest of forward compatibility, a correct user logic implementation shall ignore such records.

3.4. Loopback Stream

The loopback stream offers an easy to use and high performance data channel from the user logic to the USB 3.0 interface of the MultiHarp. Data written to the loopback stream can be read on the connected PC using the MHLib `MH_ReadFiFo()` function, just like any other T2/T3 measurement using the MultiHarp. Although the data is transferred in this way, it should not be stored in a regular PTU file, as the user defined data format may not conform to the PTU file definition.

There are 4 settings for the loopback mode. The T2 and T3 loopback modes can be used for manipulating and filtering of TTTR data streams. The user loopback mode offers the highest degree of flexibility, as arbitrary byte streams can be transferred to the host. For details on what modes are available see chapter 3.2. For details on how to set up a loopback mode see chapter 4.

The user loopback stream interface is based on the AXI-4 Stream interface. For details on the AXI-4 stream protocol see https://developer.arm.com/documentation/ihi0051/latest/. All data beats are grouped by the EFI into chunks of 128 Bytes. If a data beat has the LAST signal set, then the current group is padded with 0xA5 Bytes until it is contains 128 Bytes. The user loopback stream is not available during that time.

3.5. Loopback Stream Status Signals

The loopback stream data that is transmitted from the external FPGA into the MultiHarp is aggregated there into a FIFO. The external FPGA logic needs to ensure that the FIFO buffer never overspills. This can happen if the connected PC is not fast enough to consume data at the rate the external FPGA is producing it.

In order to prevent this, the external logic can observe the `LOOPBACK_FIFO_FILL_LEVEL` signal. It shall be interpreted as a 16 bit unsigned twos complement value. It represents the ratio of the current FIFO fill level against the maximum FIFO fill level. A value of 0 indicates the array is empty and the value 2^16-1 indicates the array is full.

The `LOOPBACK_FIFO_FILL_LEVEL` is only valid when the `LOOPBACK_FIFO_NEW_FLAGS` signal is '1'.

WARNING: The FIFO fill level is not reported to the external FPGA instantaneously, but instead represents the fill ratio at the time it was measured in the MultiHarp. The user logic must therefore take into account the data that is in flight between the external FPGA and the MultiHarp at the time of measurement. It is therefore strongly recommended to leave a safety margin of at least 2% in the FIFO.

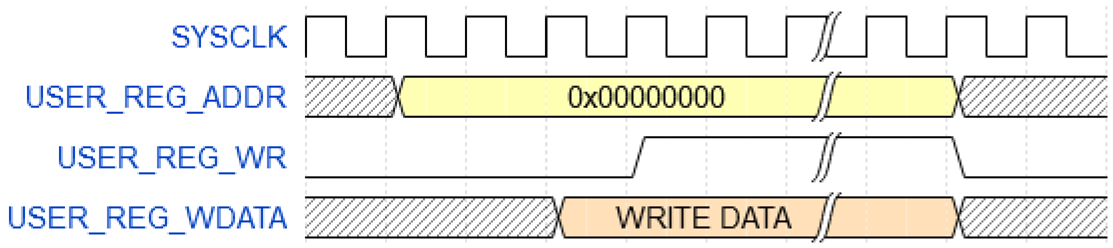
3.6. SYSCLK Domain Status Signals

The system_link_status_rec record encodes error and status of the 9 serial links. These signals are always valid and only presented to the user logic for debugging purposes. For details refer to the definition of the record in the extfpga_lib_main package.

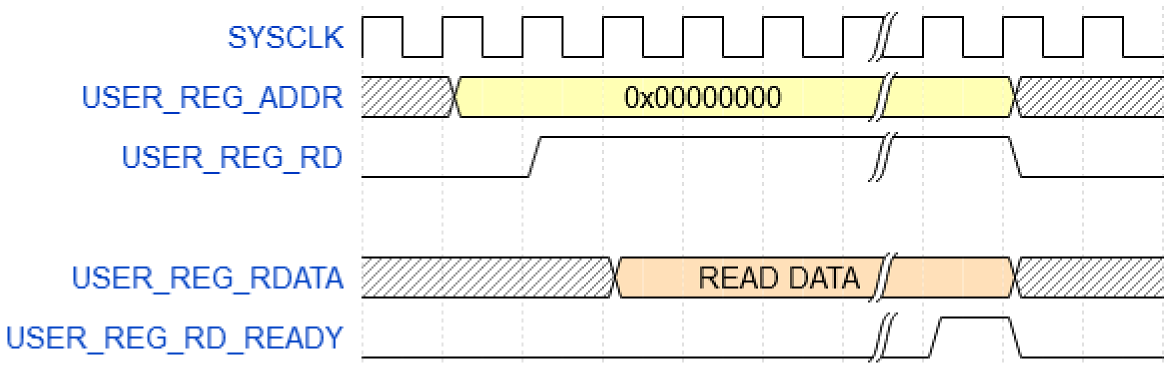
3.7. USER_REG Interface

The `USER_REG` interface is a 32 bit wide register interface for configuration purposes. It has an address space of 2^31 32 bit words from address zero upwards.

For a write operation the EFI sets the `USER_REG_ADDR` and `USER_REG_WDATA` to the desired values and asserts `USER_REG_WR`. The address and data may be set ahead of time or on the same cycle as `USER_REG_WR`. For a single write the `USER_REG_ADDR` may be asserted for more than one cycle. See the following figure for a valid example write transaction:



For a read operation the EFI sets the `USER_REG_ADDR` to the desired value and asserts `USER_REG_RD`. The address may be set ahead of time or on the same cycle as `USER_REG_RD`. The user logic must then set the `USER_REG_RDATA` to the desired value and assert `USER_RD_READY`. The returned data can be set ahead of time or on the same cycle as `USER_RD_READY`. For a single transaction the `USER_RD_READY` may only be set for one cycle. The following figure shows a valid example read transaction:



If too much time passes between the start and the completion of transaction, then the PC driver may terminate the transaction and raise an error. The user logic should therefore attempt to complete transactions as fast as possible.

3.8. I/O Signals

The EFI contains several signals that refer to physical BELs and pins in the FPGA. Those signals should be passed through to the top level entity and constrained using XDC commands. The example project contains the XDC commands for the Genesys 2 board in the file `<package location>/gateware/hdl_src/constrs/ext_genesys2_r01/ext_genesys2_pin.xdc`. For other FPGAs the correct commands will be different.

4. Software Programming Guide

The External FPGA Interface adds 5 new functions to the MHLib. Those functions can used at any point after initialization of the MultiHarp. This chapter contains a documentation for the C/C++ functions only. For instructions on how to use them in another language, please consult the MHLib documentation.

`int MH_ExtFPGAInitLink (int devidx, int linknumber, int on);`

```
arguments:          devidx:          device index 0..7
                    linknumber:      index 0..8 of the link to be initialized
                    on:              0 = off, 1 = on

return value:       =0               success
                    <0               error
```

Note: Sets the state of a link to the external FPGA for a specific device. The MultiHarp 160 base unit contains the links zero to two and every expansion unit adds two links.

`int MH_ExtFPGAGetLinkStatus (int devidx, int linknumber, unsigned int* status);`

```
arguments:          devidx:          device index 0..7
                    linknumber:      index 0..8 of the link to be queried
                    status:          pointer to unsigned int buffer of at least 9 elements

return value:       =0               success
                    <0               error
```

Note: Gets the status of a link to the external FPGA on a specific device. The MultiHarp 160 base unit contains the links zero to two and every expansion unit adds two links. For details look at the `SYSLINK_STATUS` VHDL type.

`int MH_ExtFPGASetMode (int devidx, int mode, int loopback);`

```
arguments:          devidx:          device index 0..7
                    mode:            stream mode code to be set, see mhdefin.h
                    loopback:        loopback mode code to be set, see mhdefin.h

return value:       =0               success
                    <0               error
```

Note: For details on the meaning of the mode and loopback values see chapter 3.1.

`int MH_ExtFPGAResetStreamFifos (int devidx);`

```
arguments:          devidx:          device index 0..7

return value:       =0               success
                    <0               error
```

Note: This function should typically be called after each call of the MH_Initialize() function. Calling this function is only required when using the T2DM-Mode.

`int MH_ExtFPGAUserCommand (int devidx, int write, unsigned int addr, unsigned int* data);`

```
arguments:          devidx:          device index 0..7
                    write:           0 = read, 1 = write
                    addr:            an "address" for the data in the external FPGA
                    data:            pointer to location of data to write or to receive

return value:       =0               success
                    <0               error
```

Note: Writes data to the user register at addr or reads the register contents from addr into data. For details see chapter 3.7.

5. Glossary

AXI (Advanced eXtensible Interface):

A bus protocol.

DLL (Dynamic Link Library):

A shared library system used by Microsoft Windows.

EFI (External FPGA Interface):

The PicoQuant interface for connecting TCSPC electronics to external FPGAs.

ELF (Executable and Linkable Format):

A file format for executables.

FIFO (First-In First-Out):

A type of memory queue.

FMC (FPGA Mezzanine Card):

An expansion port standard for FPGA development boards.

FPGA (Field-programmable gate array):

A type of semiconductor device that is reprogrammable.

FWHM (Full width at half maximum):

A characteristic of the measurement of a distribution.

IP (Intellectual Property):

In the context of FPGAs this refers to encapsulated pieces of digital logic used for development.

USB (Universal Serial Bus):

A peripheral standard.

TCSPC (Time-correlated single photon counting):

The measurement and analysis of arrival times of individual photons.

TTTR (Time-tagged time-resolved):

A method for storing the arrival times of photons in TCSPC measurements. Specific file formats are T2, T2DM and T3.

T2DM (TTTR2 Direct Mode):

The TTTR format for low-latency high-throughput connections such as the EFI.

VHDL (Very High Speed Integrated Circuit Hardware Description Language):

A Hardware Description Language.

XDC (Xilinx Desing Contraints File):

A file format for describing the pinout and other IO properties of Xilinx FPGAs.

6. Legal Terms

6.1. Copyright

6.2. Trademarks

This page was intentionally left blank