Video over IP FEC Receiver v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG207 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Appendix C: Debugging

Appendix D: Additional Resources and Legal Notices and Legal Notices

Introduction

The Xilinx® LogiCORE™ IP Video over IP FEC Receiver is a broadcast application module that recovers lost packets from RTP encapsulated payloads using SMPTE 2022 Forward Error Correction method. It is capable of handling a large number of video streams and is suited for deploying in 1 Gb/s networks. This core is used for developing Internet Protocol-based systems that reduce the overall cost of distribution and routing of audio and video data.

Features

- 256 or 512 channels supported
- SMPTE 2022-1 based FEC recovery
- Per channel enable
- Packet bypass per channel
- FEC recovery per channel
- Statistic counters per channel
  - Valid packets
  - Unrecoverable packets
  - Recovered packets
  - Duplicate packets
  - Reordered packets
  - Out-of-buffer packets

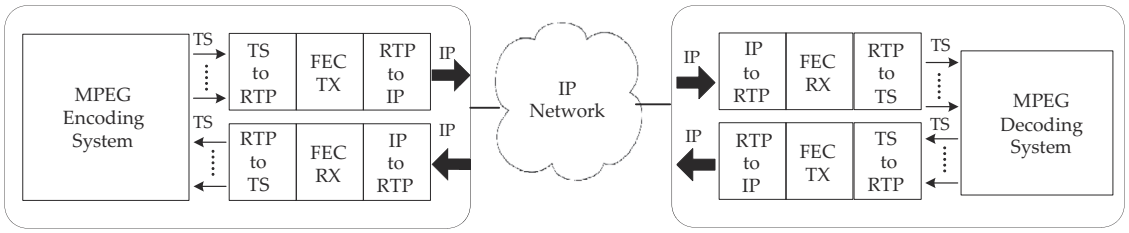
LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families, Zynq®-7000, Virtex®-7, Kintex®-7, Artix®-7
Supported User Interfaces	AXI4-Lite, AXI4-Stream, AXI-4
Resources	See Table 2-1 to Table 2-4.
Provided with Core	
Design Files	Encrypted HDL
Example Design	System Verilog
Test Bench	System Verilog
Constraints File	XDC
Simulation Model	Encrypted RTL
Supported S/W Driver(2)	N/A
Tested Design Flows(3)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. Standalone driver details can be found in the SDK directory (<*install_directory*>/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from the Xilinx Wiki page.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

As broadcast and communications markets converge, and the use of IP networks for transport of video streams becomes more attractive to broadcasters and telecommunication companies alike, the adoption of Ethernet for the transmission of multiple compressed media streams is becoming a major customer requirement. The industry is primarily looking at the SMPTE 2022 set of standards to create an open and interoperable way of connecting video over Ethernet equipment together and ensuring that Quality of Service (QoS) is high and packet loss is kept to a minimum or recovered through FEC. As shown in Figure 1-1, Video over IP FEC currently aimed at multiple transport streams carried over 1GbE networks.



SMPTE 2022-1,2
Multiple compressed streams
(typically with DVB-ASI)

Figure 1-1: **Video over IP FEC usage in either contribution and distribution networks**

The core includes Forward Error Correction (FEC). FEC protects the transport streams during the transport over IP networks. With FEC, the transmitter adds systematically generated redundant data. This carefully designed redundancy allows the receiver to detect and correct a limited number of packet errors occurring anywhere in the video without the need to ask the transmitter for additional video data. These errors, in the form of lost video packets, can be caused by many reasons, from thermal noise to storage system defects and transmission noise introduced by the environment. FEC gives the receiver the ability to correct these errors without needing a reverse channel to request retransmission of data. In real time systems, the latency is too great to request a retransmission. The ability of Xilinx FPGAs to bridge the broadcast and the communications industries by providing a modular device with highly integrable interfaces helps broadcasters reduce costs as well as reduce the overall time it takes to acquire, edit and produce content. Now that video can be reliably delivered over 1 Gbp/s Ethernet, broadcasters can replace some of the expensive mobile infrastructures supporting outside live broadcasts, as well as enable remote production from existing fixed studio set ups, which dramatically reduces both capital expenditure and operating expenses.

Feature Summary

The core is a FEC recovery module that takes in SMPTE 2022 RTP encapsulated packets. It supports both SMPTE 2022-1 and -2 standards. The core input and output are AXI4-stream interfaces. Connection to the memory is via AXI4 while the core registers are accessed using AXI4-lite interface. There are statistics counters that collect the status of the packets coming in and out for the core.

Applications

- Transport compressed constant bit rate MPEG-2 transport streams over IP networks.
- Support real-time audio/video applications in primary distribution networks.

Licensing and Ordering Information

This Xilinx LogiCORE™ IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado® Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the

core. Contact your local Xilinx sales representative for information about pricing and availability. For more information, visit the Video over IP FEC Receiver product web page. Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Product Specification

Architecture Overview

Figure 2-1 shows an overview of the SMPTE 2022-1/2 Video over IP FEC Receiver core architecture.

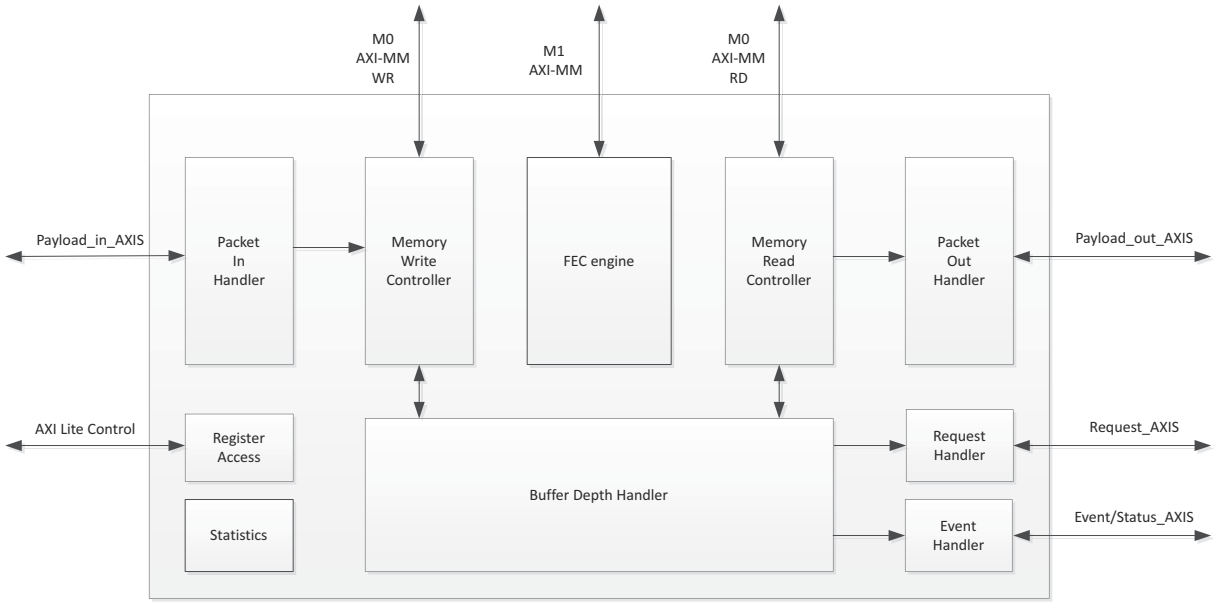


Figure 2-1: **Architecture Overview of Video over IP FEC Receiver**

The main functional blocks of the core are:

Packet in handler: Demultiplex the Ethernet packets into channel streams.

Memory write controller: Puts Ethernet packets into the DDR memory buffer.

Memory read controller: Retrieves Ethernet packets from the DDR memory buffer.

Packet out handler: Convert the payload of the packets into AXI-Stream format

Buffer depth handler: Update the buffer level of the packets in the DDR.

Request handler: Takes in the request for packets or status from the user.

Event handler: Sends out events of packets arrival or status to the user

FEC engine: Performs FEC recovery on packet loss in the DDR memory buffer.

Register access: Register configuration and status read-back on the core.

Statistics counters

Standards

The SMPTE 2022-1/2 Video over IP Receiver core is compliant with the AXI4, AXI4-Stream and AXI4-Lite interconnect standards. See the Video IP: AXI Feature Adoption section of the Vivado AXI Reference Guide (UG1037)[Ref 5] for additional information.

The function of the core is compliant with SMPTE 2022-1 and SMPTE 2022-2 standards.

Performance

Maximum Frequencies

The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the FPGA device, different versions of Xilinx® tools, and other factors. See Table 2-1 through Table 2-4 for device family–specific information.

Resource Utilization

Resources required for this core have been estimated for Zynq®-7000, Virtex-7®, Kintex-7®, and Artix®-7 devices. UltraScale™ results are expected to be similar to 7 series results. These values were generated using Xilinx Vivado® Design Suite. They are derived from post-synthesis reports, and might change during MAP and PAR.

Table 2-1: **Resource Utilization for Artix-7 FPGAs**

Part Information			Performance and Resource Utilization							
Speed Grade	C_NUM_OF_CHANNELS	Fixed clocks (MHz)	Clock Input	Fmax (MHz)	LUTs	FFs	LUT-FF Pairs	DSP48s	36k BRAMs	18k BRAMs
-1	256	s_axi_aclk=100	core_clk	150	3684	6908	6029	0	37	5
-2	256	s_axi_aclk=100	core_clk	150	3667	6908	6039	0	37	5
-1	512	s_axi_aclk=100	core_clk	150	3682	6927	6038	0	37	5
-2	512	s_axi_aclk=100	core_clk	150	3669	6927	6056	0	37	5
Device: xc7a200t Package: sbg484 Speedfile: PRODUCTION 1.14 2014-09-11										

Table 2-2: **Resource Utilization for Kintex-7 FPGAs**

Part Information			Performance and Resource Utilization							
Speed Grade	C_NUM_OF_CHANNELS	Fixed clocks (MHz)	Clock Input	Fmax (MHz)	LUTs	FFs	LUT-FF Pairs	DSP48s	36k BRAMs	18k BRAMs
-1	256	s_axi_aclk=100	core_clk	150	3772	6908	5978	0	37	5
-2	256	s_axi_aclk=100	core_clk	150	3856	6908	6117	0	37	5
-1	512	s_axi_aclk=100	core_clk	150	3769	6927	6252	0	37	5
-2	512	s_axi_aclk=100	core_clk	150	3850	6927	6020	0	37	5
Device: xc7k480t Package: ffg901 Speedfile: PRODUCTION 1.12 2014-09-11										

Table 2-3: **Resource Utilization for Virtex-7 FPGAs**

Part Information			Performance and Resource Utilization							
Speed Grade	C_NUM_OF_CHANNELS	Fixed clocks (MHz)	Clock Input	Fmax (MHz)	LUTs	FFs	LUT-FF Pairs	DSP48s	36k BRAMs	18k BRAMs
-1	256	s_axi_aclk=100	core_clk	150	3775	6908	6199	0	37	5
-2	256	s_axi_aclk=100	core_clk	150	3846	6908	6172	0	37	5
-1	512	s_axi_aclk=100	core_clk	150	3768	6927	5902	0	37	5
-2	512	s_axi_aclk=100	core_clk	150	4046	6927	6428	0	37	5
Device: xc7vx330t Package: ffg1157 Speedfile: PRODUCTION 1.12 2014-09-11										

Table 2-4: **Resource Utilization for Zynq-7000 AP SoC FPGAs**

Part Information			Performance and Resource Utilization							
Speed Grade	C_NUM_OF_CHANNELS	Fixed clocks (MHz)	Clock Input	Fmax (MHz)	LUTs	FFs	LUT-FF Pairs	DSP48s	36k BRAMs	18k BRAMs
-1	256	s_axi_aclk=100	core_clk	150	3777	6908	5976	0	37	5
-2	256	s_axi_aclk=100	core_clk	150	4044	6908	6187	0	37	5
-1	512	s_axi_aclk=100	core_clk	150	3776	6927	5956	0	37	5
-2	512	s_axi_aclk=100	core_clk	150	3841	6927	6170	0	37	5
Device: xc7z100 Package: ffg900 Speedfile: PRODUCTION 1.11 2014-09-11										

Resource Utilization is calculated using `core_clk` and the frequency of other clock fixed at `s_axi_aclk`=100 MHz. The maximum clock frequency results were obtained by double-registering input and output ports to reduce dependence on I/O placement. The inner level of registers used a separate clock signal to measure the path from the input registers to the first output register through the core. The results are post-implementation, using tool default settings except for high effort.

The resource usage results do not include the "characterization" registers and represent the true logic used by the core. LUT counts include SRL16s or SRL32s.

Clock frequency does not take clock jitter into account and should be de rated by an amount appropriate to the clock source jitter specification. The maximum achievable clock frequency and the resource counts might also be affected by other tool options, additional logic in the FPGA, using a different version of Xilinx tools, and other factors.

Port Descriptions

The Video over IP FEC Receiver core uses industry-standard control and data interfaces to connect to other system components. The following sections describe the various interfaces available with the core. Figure 2-2 provides an I/O diagram of the core.

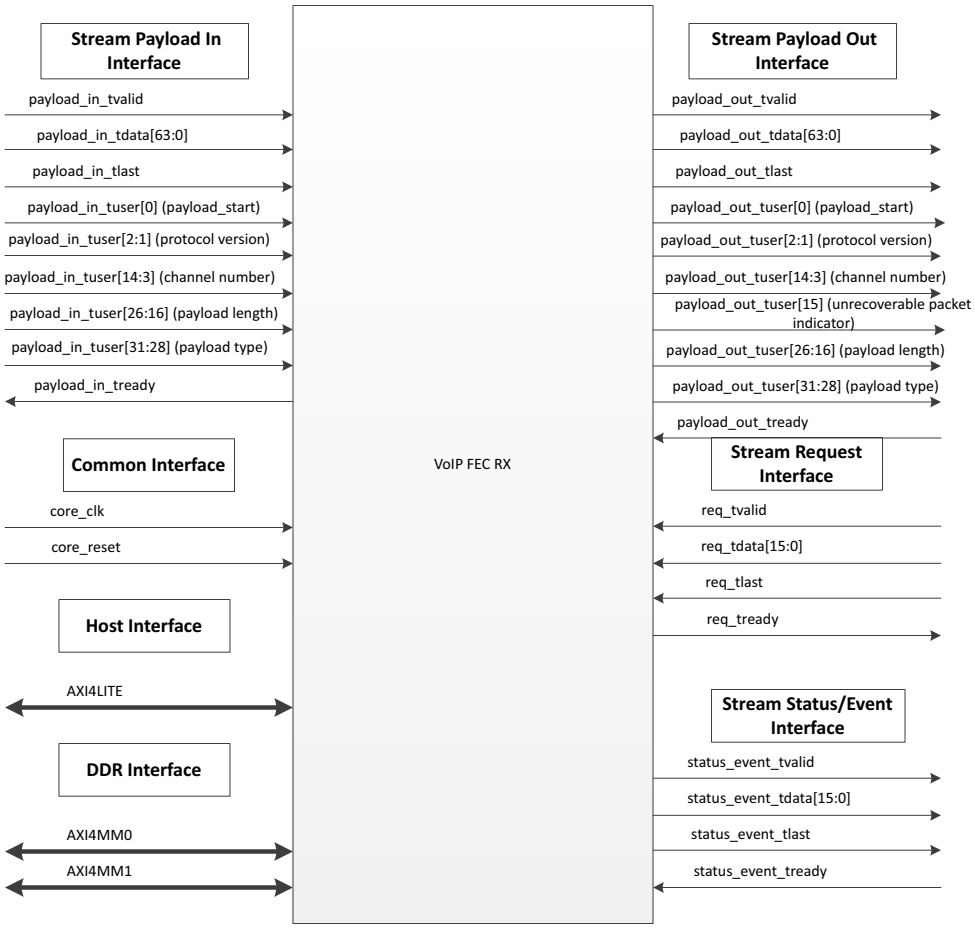


Figure 2-2: **Core I/O**

Figure 2-3 shows the core in IP integrator.

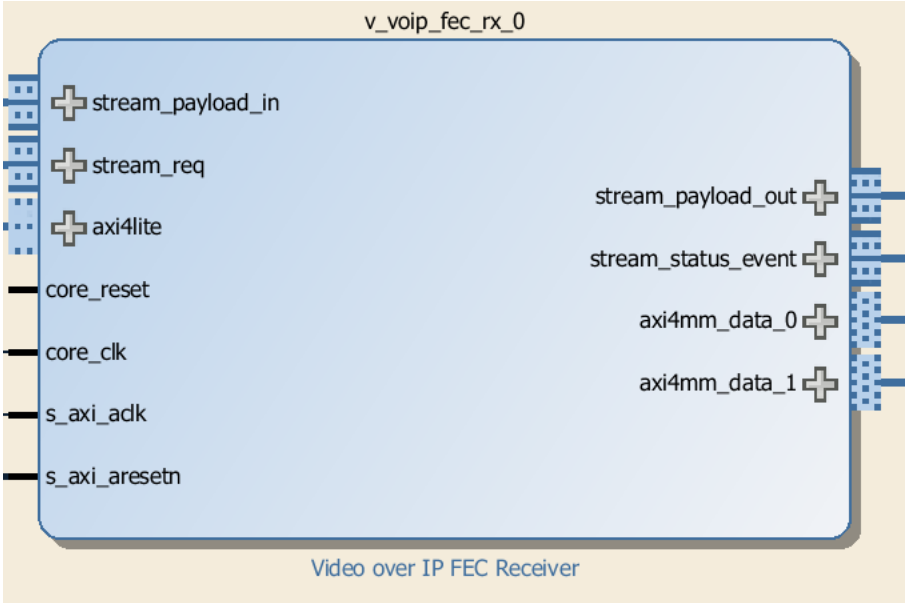


Figure 2-3: **IP Integrator I/O**

Table 2-5: **Mapping from IP Integrator to Core**

IP integrator I/O	Core I/O
stream_payload_in	stream payload in interface
stream_req	stream request interface
axi4lite	host interface
core_reset	core_reset
core_clk	core_clk
s_axi_aclk	host interface
s_axi_aresetn	host interface
stream_payload_out	Stream payload out interface
stream_status_event	Stream status/event
axi4mm_data_0	axi4mm0
axi4mm_data_1	axi4mm1

Common Interface

Table 2-6 describes the common signals.

Table 2-6: **Common Signals Description**

Signals	Direction	Width	Description
core_clock	In	1	The main operating clock for the core. It applies to all the interfaces except the host.
core_reset	In	1	The main reset for the core. It applies to payload_in, payload_out, stream_request and stream_event interfaces. It is synchronous to core_clock and is active high.

Stream Payload In Interface

Table 2-7 describes the stream payload.

Table 2-7: **Stream Payload in Signals**

Signals	Direction	Width	Description
payload_in_tvalid	In	1	High from the start to the end of the packet transfer.
payload_in_tdata	In	64	Data
payload_in_tlast	In	1	High at the last word of the input packet
payload_in_tuser	In	32	[0]: Payload start. High at the first valid word of the input packet. [2:1]: Protocol version (set to “00”) [14:3]: Channel number. Valid at payload start. [15]: Reserved [26:16]: Payload length. Valid at payload start. [27]: Reserved [31:28]: Payload type. Valid at payload start. “0000” – UDP packet “0001” – SMPTE 2022-2 compliant RTP packet “0010” – SMPTE 2022-1 compliant Column FEC packet “0011” – SMPTE 2022-1 compliant Row FEC packet
payload_in_tready	Out		May be asserted low in between packet transfers.

Note: First packet byte is `payload_in_tdata [7:0]` of the first word transfer. Able to accept approximately 2KB per packet transfer.

Stream Payload Out Interface

Table 2-8 describes the stream payload.

Table 2-8: **Stream Payload Out Signals**

Signals	Direction	Width	Description
payload_out_tvalid	Out	1	High from the start to the end of the packet transfer.
payload_out_tdata	Out	64	Data
payload_out_tlast	Out	1	High at the last word of the output packet
payload_out_tuser	Out	32	[0]: Payload start. High at the first valid word of the output packet. [2:1]: Protocol version (set to "00") [14:3]: Channel number. Valid at payload start. [15]: Unrecoverable packet indicator. High if packet is not available. Valid at payload start. [26:16]: Payload length. Valid at payload start. [27]: Reserved [31:28]: Payload type. Valid at payload start. "0000" – UDP packet "0001"–SMPTE2022-2compliantRTPpacket
payload_out_tready	In	1	May be asserted low in between packet transfers.

Stream Request Interface

Table 2-9: **Stream Request Signals**

Signals	Direction	Width	Description
req_tvalid	In	1	High from the start to the end of the request.
req_tdata	In	16	Data
req_tlast	In	1	High at the last word of the request
req_tready	Out	1	May be asserted low in between requests.

The `req_tdata` signal contains REQ_ID and channel number.

Table 2-10: **Stream Request ID**

Req ID	Request	Description
1	Payload	Request packet for a particular channel.
2	Buffer depth	Request buffer depth of a particular channel.

Stream Event Interface

Table 2-11: **Stream Event Signals Description**

Signals	Direction	Width	Description
status_event_tvalid	Out	1	High from the start to the end of the status/event.
status_event_tdata	Out	16	Data
status_event_tlast	Out	1	High at the last word of the status/event.
status_event_tready	In	1	May be asserted low in between events.

The `status_event_tdata` signal contains EVENT ID, channel number, buffer depth, FEC L and FEC D values.

Table 2-12: **Stream Event ID**

Event ID	Event	Description
1	Received Packet	Generated when a packet is received for a particular channel.
2	Empty Buffer	Generated when a request is received for particular channel, but no packets in the buffer for that channel.
3	Channel Status	Generated when a request is received on enquiring buffer depth of a particular channel (Req ID 2).

Memory Interface (AXI4MM0 and AXI4MM1)

The memory interface uses an AXI4 interface to connect to an AXI4 interconnect which provides the access to the external memory through an AXI4 DDR controller. The AXI4 interface data width is 256. See the *LogiCORE IP AXI Interconnect Product Guide* (PG059) for more information.

Table 2-13: **AXI4 Memory Interface Signals**

Signal Name	Direction	Width	Description
m0_axi_awid	Out	1	Write Address Channel Transaction ID.
m0_axi_awaddr	Out	32	Write Address Channel Address.
m0_axi_awlen	Out	8	Write Address Channel Burst Length code.
m0_axi_awsize	Out	3	Write Address Channel Transfer Size code.
m0_axi_awburst	Out	2	Write Address Channel Burst Type.
m0_axi_awlock	Out	2	Write Address Channel Atomic Access Type.
m0_axi_awcache	Out	4	Write Address Channel Cache Characteristics.
m0_axi_awprot	Out	3	Write Address Channel Protection Bits.
m0_axi_awqos	Out	4	Write Address Channel Quality of Service.
m0_axi_awvalid	Out	1	Write Address Channel Valid.

Table 2-13: **AXI4 Memory Interface Signals *(Cont'd)***

Signal Name	Direction	Width	Description
m0_axi_awready	In	1	Write Address Channel Ready.
m0_axi_wdata	Out	256	Write Data Channel Data.
m0_axi_wstrb	Out	32	Write Data Channel Data Byte Strobes.
m0_axi_wlast	Out	1	Write Data Channel Last Data Beat.
m0_axi_wvalid	Out	1	Write Data Channel Valid.
m0_axi_wready	In	1	Write Data Channel Ready.
m0_axi_bid	In	1	Write Response Channel Transaction ID.
m0_axi_bresp	In	2	Write Response Channel Response Code.
m0_axi_bvalid	In	1	Write Response Channel Valid.
m0_axis_bready	Out	1	Write Response Channel Ready.
m0_axi_arid	Out	1	Read Address Channel Transaction ID.
m0_axi_araddr	Out	32	Read Address Channel Address
m0_axi_arlen	Out	8	Read Address Channel Burst Length code.
m0_axi_arsize	Out	3	Read Address Channel Transfer Size code.
m0_axi_arburst	Out	2	Read Address Channel Burst Type.
m0_axi_arlock	Out	2	Read Address Channel Atomic Access Type.
m0_axi_arcache	Out	4	Read Address Channel Cache Characteristics.
m0_axi_arprot	Out	3	Read Address Channel Protection Bits.
m0_axi_arqos	Out	4	AXI4 Read Address Channel Quality of Service.
m0_axi_arvalid	Out	1	Read Address Channel Valid.
m0_axi_arready	In	1	Read Address Channel Ready.
m0_axi_rid	In	1	Read Data Channel Data Transaction ID.
m0_axi_rdata	In	256	Read Data Channel Data.
m0_axi_rresp	In	2	Read Data Channel Response Code.
m0_axi_rlast	In	1	Read Data Channel Last Data Beat.
m0_axi_rvalid	In	1	Read Data Channel Valid.
m0_axi_rready	Out	1	Read Data Channel Ready.
m1_axi_awid	Out	1	Write Address Channel Transaction ID.
m1_axi_awaddr	Out	32	Write Address Channel Address.
m1_axi_awlen	Out	8	Write Address Channel Burst Length code.

Table 2-13: **AXI4 Memory Interface Signals *(Cont'd)***

Signal Name	Direction	Width	Description
m1_axi_awsize	Out	3	Write Address Channel Transfer Size code.
m1_axi_awburst	Out	2	Write Address Channel Burst Type.
m1_axi_awlock	Out	2	Write Address Channel Atomic Access Type.
m1_axi_awcache	Out	4	Write Address Channel Cache Characteristics.
m1_axi_awprot	Out	3	Write Address Channel Protection Bits.
m1_axi_awqos	Out	4	Write Address Channel Quality of Service.
m1_axi_awvalid	Out	1	Write Address Channel Valid.
m1_axi_awready	In	1	Write Address Channel Ready.
m1_axi_wdata	Out	256	Write Data Channel Data.
m1_axi_wstrb	Out	32	Write Data Channel Data Byte Strobes.
m1_axi_wlast	Out	1	Write Data Channel Last Data Beat.
m1_axi_wvalid	Out	1	Write Data Channel Valid.
m1_axi_wready	In	1	Write Data Channel Ready.
m1_axi_bid	In	1	Write Response Channel Transaction ID.
m1_axi_bresp	In	2	Write Response Channel Response Code.
m1_axi_bvalid	In	1	Write Response Channel Valid.
m1_axis_bready	Out	1	Write Response Channel Ready.
m1_axi_arid	Out	1	Read Address Channel Transaction ID.
m1_axi_araddr	Out	32	Read Address Channel Address
m1_axi_arlen	Out	8	Read Address Channel Burst Length code.
m1_axi_arsize	Out	3	Read Address Channel Transfer Size code.
m1_axi_arburst	Out	2	Read Address Channel Burst Type.
m1_axi_arlock	Out	2	Read Address Channel Atomic Access Type.
m1_axi_arcache	Out	4	Read Address Channel Cache Characteristics.
m1_axi_arprot	Out	3	Read Address Channel Protection Bits.
m1_axi_arqos	Out	4	AXI4 Read Address Channel Quality of Service.
m1_axi_arvalid	In	1	Read Address Channel Valid.
m1_axi_arready	In	1	Read Address Channel Ready.
m1_axi_rid	In	1	Read Data Channel Data Transaction ID.
m1_axi_rdata	In	256	Read Data Channel Data.

Table 2-13: **AXI4 Memory Interface Signals *(Cont'd)***

Signal Name	Direction	Width	Description
m1_axi_rresp	In	2	Read Data Channel Response Code.
m1_axi_rlast	In	1	Read Data Channel Last Data Beat.
m1_axi_rvalid	In	1	Read Data Channel Valid.
m1_axi_rready	Out	1	Read Data Channel Ready.

Host Interface (AXI4LITE)

The host interface allows user to control core parameters. Core configuration can be done using an embedded ARM or soft system processor such as MicroBlaze. This AXI4-Lite slave interface facilitates integrating the core into a processor system, or along with other video or AXI4-Lite compliant IP, connected via AXI4-Lite interface to an AXI4-Lite master. See the *LogiCORE IP AXI Interconnect Product Guide* (PG059) for more information.

Table 2-14: **Host Interface Signals**

Signal Name	Direction	Width	Description
s_axi_clk	In	1	AXI4-Lite Clock.
s_axi_aresetn	In	1	AXI4-Lite Active-Low Reset.
s_axi_awaddr	In	9	AXI4-Lite Write Address Bus
s_axi_awvalid	In	1	AXI4-Lite Write Address Channel Write Address Valid.
s_axi_wdata	In	32	AXI4-Lite Write Data Bus
s_axi_wstrb	In	4	AXI4-Lite Write Data Channel Data Byte Strobes.
s_axi_wvalid	In	1	AXI4-Lite Write Data Channel Write Data Valid.
s_axi_awready	Out	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates DMA ready to accept the write address.
s_axi_wready	Out	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates DMA is ready to accept the write data.
s_axi_bresp	Out	2	AXI4-Lite Write Response Channel. Indicates results of the write transfer.
s_axi_bvalid	Out	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid.
s_axi_bready	In	1	AXI4-Lite Write Response Channel Ready. Indicates target is ready to receive response.
s_axi_arvalid	In	1	AXI4-Lite Read Address Channel Read Address Valid
s_axi_arready	Out	1	Ready. Indicates DMA is ready to accept the read address.
s_axi_araddr	In	9	AXI4-Lite Read Address Bus

Table 2-14: **Host Interface Signals *(Cont'd)***

Signal Name	Direction	Width	Description
s_axi_rready	In	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates target is ready to accept the read data.
s_axi_rdata	Out	32	AXI4-Lite Read Data Bus
s_axi_rresp	Out	2	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer.
s_axi_rvalid	Out	1	AXI4-Lite Read Data Channel Read Data Valid

Register Space

The registers are categorized into two sections, the general space and channel space. The registers in the general space apply to all the channels while the channel space registers apply only to the specific channel set by register offset 0x08.

The general registers can be access through normal address read and write.

To configure a channel, observe the following steps.

1. Before configuring a channel, check the busy bit (bit 0, register offset 0x04) to be low.
2. Set the channel to be configured at register offset, 0x08.
3. Configure the channel specific registers.
4. Pulse channel update bit (bit 1, register offset 0x00) to commit the channel parameters.
5. Repeat steps 1-4 for another channel. See Figure 2-4.

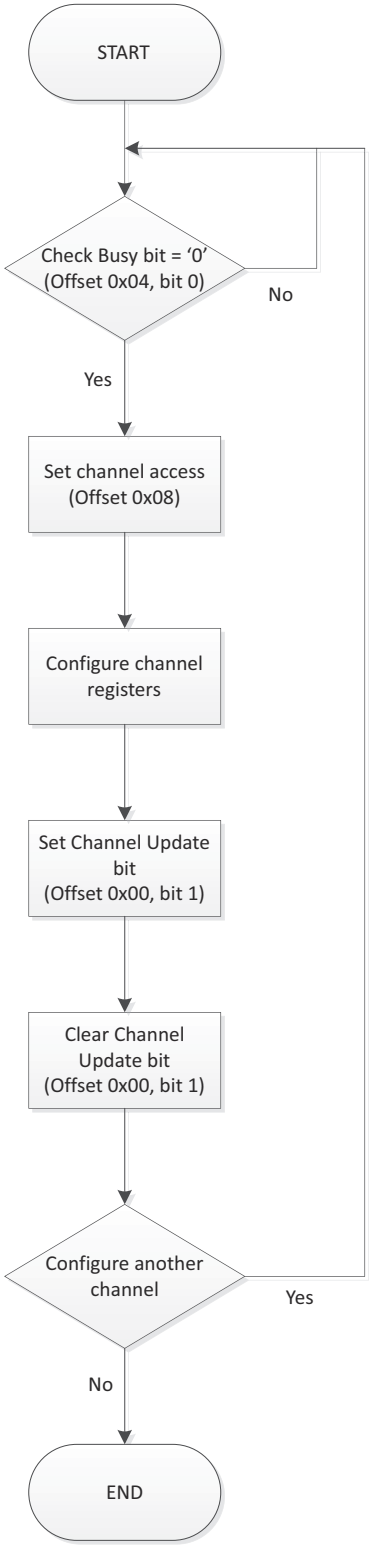


Figure 2-4: **Configuration of Channel Registers Flowchart**

To read off a channel register, set the channel access at offset 0x08 before proceeding.

Table 2-15 describes the core registers.

Table 2-15: **Registers**

Address Offset (HEX)	Register Name	Access Type	Default Value (HEX)	Register Description	
				Bit Range	Value
GENERAL					
0x000	control	R/W	0x00000000	**Control**	
				31:3	Reserved
				2	**Clear channel** Reset the channel on Channel Access at offset 0x08.
				1	**Channel update** Allows configured registers to take effect for the channel on Channel Access at offset 0x08.
				0	Reserved. Set at 0.
0x004	status	R		**Status**	
				31:1	Reserved
				0	**Update Busy** 1 - Core busy updating configured parameters or clearing internal buffer 0 - Core able to accept new configuration
0x008	channel_access	R/W	0x00000000	**Channel access**	
				31:12	Reserved
				11:0	The channel number to access registers
0x00C	sys_config	R		**System configuration**	
				31:17	Reserved
				16	FEC recovery supported
				15:0	Number of channels supported

Table 2-15: **Registers *(Cont'd)***

Address Offset (HEX)	Register Name	Access Type	Default Value (HEX)	Register Description	
				Bit Range	Value
0x010	version	R	0x01000000	**Hardware Version**	
				31:24	Version major
				23:16	Version minor
				15:12	Version revision
				11:8	Patch ID
				7:0	Revision number
0x020	fec_processing_delay	R/W	0x00000000	**FEC Processing Delay**	
				31:0	Time delay for the incoming FEC packets to be processed. (Based on core_clock ticks)
CHANNEL					
0x080	chan_conf	R/W	0x00000000	**Channel Configuration**	
				31:3	Reserved
				2	**FEC recovery disable** 1 - FEC recovery off 0 - FEC recovery on
				1	**Media packet bypass** 1 - Media packet is First In First Out 0 - Media packet is being reordered
				0	**Channel enable** 1 - enable channel 0 - disable channel.
0x090	valid pkts_cnt	R		**Valid Packet Count**	
				31:0	Number of valid packets received in the channel
0x094	unrecv_ pkts_cnt	R		**Unrecovered Packet Count**	
				31:0	Number of missing packets in the channel
0x098	corr_ pkts_cnt	R		**Corrected Packet Count**	
				31:0	Number of corrected packets in the channel

Table 2-15: **Registers *(Cont'd)***

Address Offset (HEX)	Register Name	Access Type	Default Value (HEX)	Register Description	
				Bit Range	Value
0x09C	dup_ pkts_cnt	R		**Duplicated Packet Count**	
				31:0	Number of duplicated packets in the channel
0x0A0	reordered_pkts_cnt	R		**Reordered Packet Count**	
				31:0	Number of reordered packets in the channel
0x0A4	oob_ pkts_cnt	R		**Out-of-buffer Packet Count**	
				31:0	Number of packets discarded when Channel Media Buffer is full.
0x0A8	channel_status	R		**Channel Status**	
				31:22	Reserved
				21	Row FEC detected
				20	Column FEC detected
				19:10	FEC D value detected
				9:0	FEC L value detected
0x0AC	curr_buffer_depth	R		**Current Buffer Depth**	
				31:17	Reserved
				16:0	Expected number of media packets buffered in the DDR
0xB0	oor_ pkts_cnt	R		**Out-of-range Packet Count**	
				31:0	Number of late coming packets that are discarded.

Register Description

Control (0x000) Register

Channel_update (bit 1) is a write-done semaphore for the host processor, which facilitates committing all user register updates in the channel space simultaneously. One set of registers (the processor registers) is directly accessed by the processor interface, while the other set (the active set) is actively used by the core. New values written to the processor registers are copied over to the active set if and only if the register update bit is set. Setting the bit to 0 before updating multiple registers and then setting the bit to 1 when updates are completed ensures all channel space registers are updated simultaneously.

An active high pulse to Clear_channel (bit 2) clear the channel to accept a new payload stream. The channel is set in Channel Access at offset 0x08.

Status (0x004) Register

The update_busy bit asserts when the core is updating the new configuration or clearing internal buffer. Do not configure the core if this bit is high.

Channel_access (0x008) Register

This register is used when accessing channel space registers. Always set the channel first as shown in Figure 2-4.

Sys_config (0x00C) Register

Readback register for user to know if the core has being implemented to support FEC recovery and the number of channels

Version (0x010) Register

Bit fields of the register facilitate software identification of the exact version of the hardware peripheral incorporated into a system. The core driver can take advantage of this read-only value to verify that the software is matched to the correct version of the hardware.

FEC_processing_delay (0x020) Register

Set time delay for incoming FEC packets before processing for recovery in order to cater scenario such as packets arriving out of order. Value is count based on core clock tick.

Chan_conf (0x080) Register

If media_packet_bypass bit is set, FEC_recovery_disable bit is ignored. Packet recovery is not performed and incoming FEC packets are discarded. Media packets are pulled out in a First In, First Out fashion.

If media_packet_bypass bit is low, FEC_recovery_disable bit can be cleared for the core to perform FEC recovery. Media packet are reordered depending on the buffer depth maintained for the channel.

Valid_pkt_cnt (0x090) Register

Valid packet count increments when a packet belonging to the channel is received. Channel register clears itself when read.

Unrecv_pkt_cnt (0x094) Register

Unrecoverable packet count increments when a media packet is missing in the channel stream. Channel register clears itself when read.

Corr_pkt_cnt (0x098) Register

Corrected packet count increments when a media packet is being recovered in the channel stream. Channel register clears itself when read.

Dup_pkt_cnt (0x09C) Register

Duplicated packet count increments when a channel received a media packet that already exists in the media buffer. This duplicated media packet is discarded. Channel register clears itself when read.

Reordered_pkt_cnt (0x0A0) Register

Reordered packet count increments when the media packet received belongs to an earlier packet. Channel register clears itself when read.

Oob_pkts_cnt (0x0A4) Register

Out-of-buffer packet count increments when a media packet is discarded due to channel buffer full. Channel register clears itself when read.

Channel_status (0x0A8) Register

This register reflects the FEC matrix size detected for the channel. Only detected FEC column packet update the FEC L and D value.

- Bit [9:0] - L value of the FEC matrix detected
- Bit [19:10] - D value of the FEC matrix detected
- Bit [20] - High indicates column FEC received
- Bit [21] - High indicates row FEC received

Curr_buffer_depth (0x0AC) Register

Current number of media packets being buffered for channel.

Oor_pkts_cnt (0x0B0) Register

Out-of-range packet count increments when an incoming media packet is discarded due to it being an earlier packet compared to the outgoing media packet from the core. Channel register clears itself when read.

Designing with the Core

Figure 3-1 shows an example of an application design using Video over IP FEC RX core with other Xilinx IP.

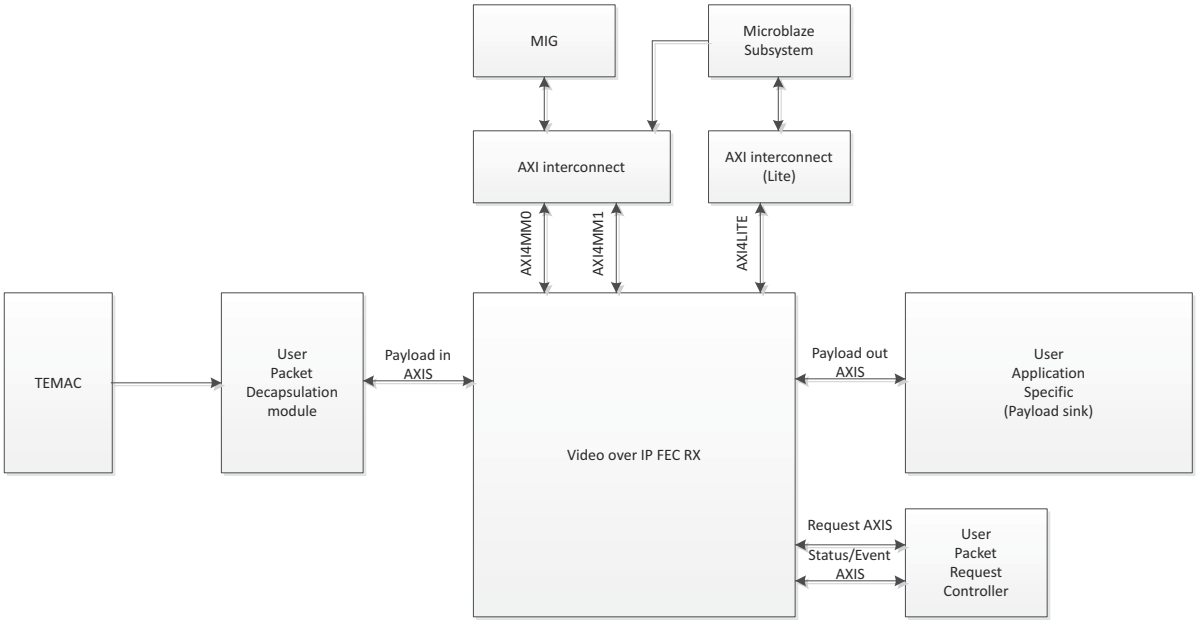


Figure 3-1: **Example of an Application Design with Video over IP FEC RX Core**

This section describes how Video over IP FEC Receiver core can be design in to build a fully functional design with user application logic.

The core accepts SMPTE 2022-1/2 encapsulated payload for FEC recovery. Figure 3-2 and Figure 3-3 show the types of packet structure for incoming data format to RX. When a packet is received from the Ethernet, a user packet decapsulation module is required to decode which stream the packet belongs to and also to strip down the packet to RTP encapsulated layer. This data is then fed to Video over IP FEC Receiver core through an AXI4-stream transaction. Figure 3-3 shows the waveform of the transaction together with the sideband packet information.

WORD\BYTE	7 downto 0	15 downto 8	23 downto 16	31 downto 24	39 downto 32	47 downto 40	55 downto 48	63 downto 56
1	V(2)/P(1)/X(1)/CC(4)	M(1)/PT(7)	sequence number		timestamp			
2	SSRC identifier							
...	Media payload							
...								
...								
...								

Figure 3-2: **Media Payload with SMPTE 2022-2 Header**

WORD\BYTE	7 downto 0	15 downto 8	23 downto 16	31 downto 24	39 downto 32	47 downto 40	55 downto 48	63 downto 56
1	V(2)/P(1)/X(1)/CC(4)	M(1)/PT(7)	sequence number		timestamp			
2	SSRC identifier				SNBase low bits		Length Recovery	
3	E/PT recovery	Mask			TS recovery			
4	ND/type/index	offset	NA	SNBase ext bits				
...	FEC payload							
...								
...								
...								

Figure 3-3: **FEC Payload with SMPTE 2022-1 Header**

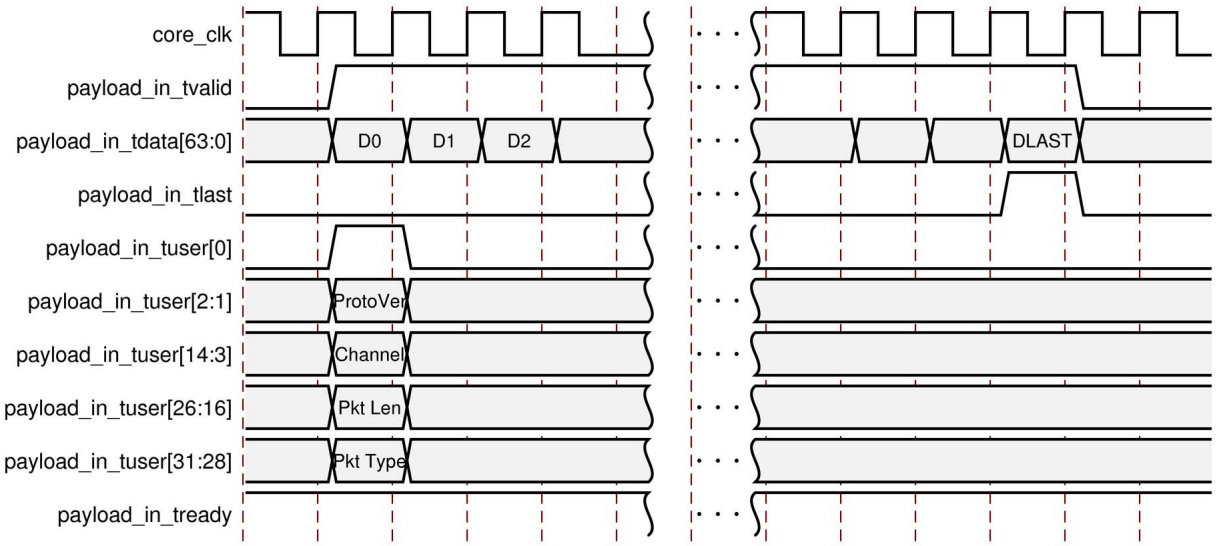


Figure 3-4: **Payload in AXI4-Stream Waveform**

The Video over IP FEC Receiver core requires access to DDR memory for normal operation and to perform FEC recovery. The core stores incoming packets into DDR and puts media and FEC payloads at different sections of the memory. The user has to set 2 base addresses in the GUI, one for the media payload buffer and the other for FEC payload buffer. Note that the memory space of media payload buffer should not overlap FEC payload buffer. The total size of the buffers is based on the number of channels selected in the GUI. The memory requirement and bandwidth consumption are given below.

Memory Requirement

Each valid incoming payload packet is allocated a fixed size of 2kB when stored in the DDR memory. For each channel stream, the RX core is able to store up to 256 packets. Based on 512 channels, DDR memory requirement for media packets is 256MB. (512 channels * 256 packets * 2kB).

Video over IP FEC Receiver core reserves 80 packet slots per channel for FEC payload storage and the memory buffer requirement is 80MB. (512 channels * 80 packets * 2kB).

Memory Bandwidth Consumption

Table 3-1: **Memory Bandwidth Consumption**

Bandwidth Utilization (Gbps)			
AXI4MM0 WRITE	AXI4MM0 READ	AXI4MM1 WRITE	AXI4MM1 READ
2	1	0.5	2

Other than the input AXI4-stream, the FEC RX core has three more AXI4-stream interfaces at the user side. The first is `payload_out` interface bus. It is symmetrical to the payload_in interface. Only media and recovered payloads come out of the `payload_out` AXI-stream interface. There is 1 signal different at the `payload_out` interface, that is, unrecoverable packet indicator. This signal is High when the outgoing packet is non-existent in the core and the payload data is all zeros. Figure 3-5 shows the waveform of an outgoing packet transaction. The output data is in the form of SMPTE 2022-2 encapsulated payload.

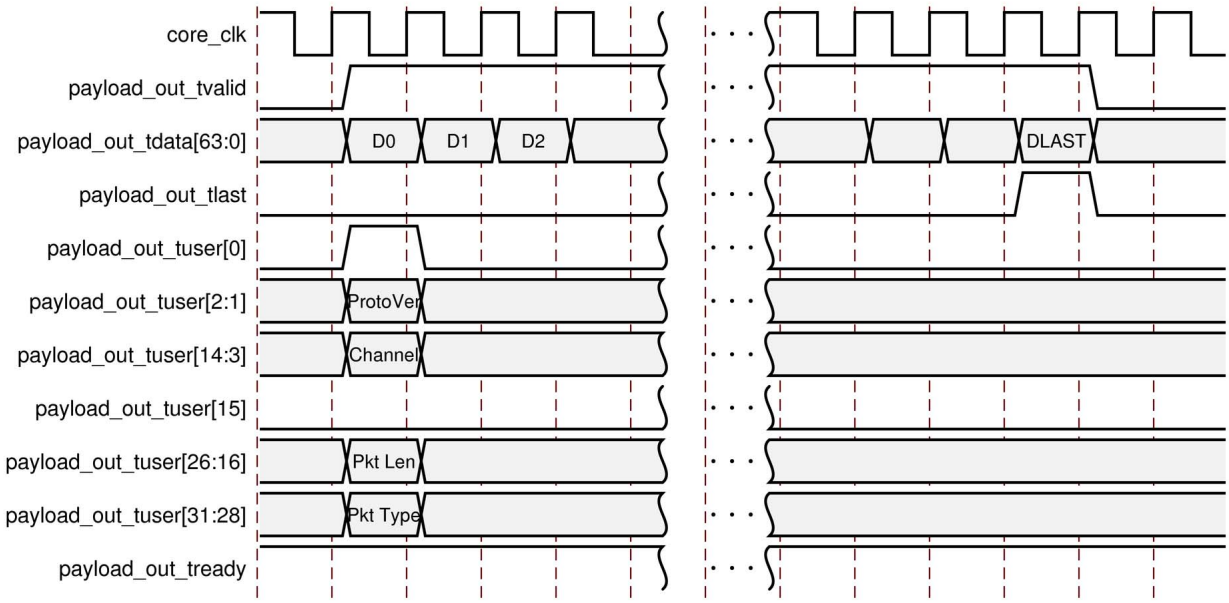


Figure 3-5: **Payload Out AXI4-Stream Waveform**

Whenever there is an incoming valid packet into the FEC RX, the core processes the packet and sends out an event through the status_event AXI4-stream interface. This event informs the user which channel has newly arrived payload, the current accumulated packets in the core for this channel, and if there is any FEC packet detected on a certain matrix size. Note that do not allow the accumulated buffer depth to go beyond 256.

Other than a received packet event as mention above, there are 2 more types of events generated by the core as described in Table 2-11. Figure 3-6 illustrates an event transaction.

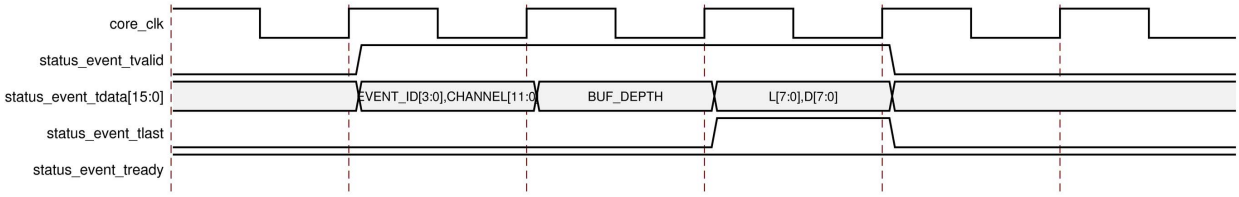


Figure 3-6: **Status/Event AXI4-Stream Waveform**

EVENT_ID: As explained in Table 2-11.

CHANNEL: Channel number of this event.

BUF_DEPTH: Current accumulated packets for the channel

L: L value for the FEC matrix size detected

D: D value for the FEC matrix size detected

At a suitable level of packets accumulated per channel, user can pull the packets out from the core via the request AXI4-stream interface. Figure 3-7 shows how this can be done and Table describes the types of request user can send to the core. Requests are served in a first in, first out basis by the core.

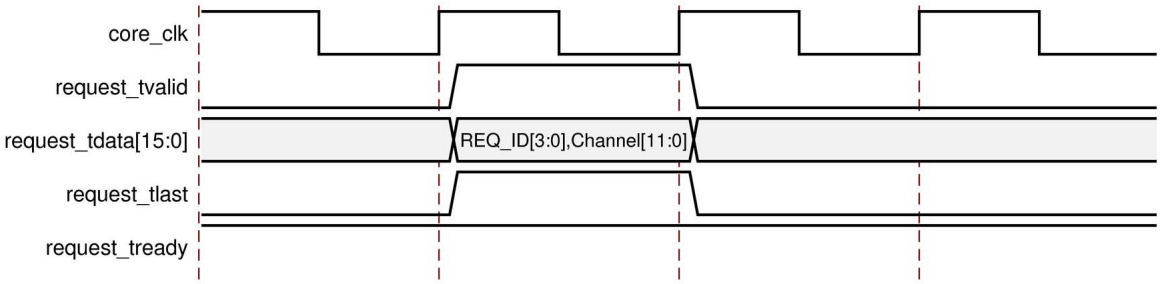


Figure 3-7: **Request AXI4-Stream Waveform**

The amount of packets buffered per channel depends on user requirement. However, to allow proper FEC recovery to be performed on loss packets, a minimum size of L*D*2 must be maintain at the buffer depth. In Figure 3-1, a user packet request controller may be built to monitor the events coming from the core and at the same time control the buffer depth level for each active channel.

User Packet Request Controller is a module to request packet out from the VoIP FEC RX core. It is recommended that user pay attention to the following points when designing this module.

1. The controller should consistently monitor the Status/Event AXI4-Stream interface for Event ID 1 that contains information on the current media buffer depth, FEC L and FEC D per channel.
2. Request packet can start when the media buffer depth becomes 1.
3. If the channel is set for FEC recovery enable, the controller should only start to request packet when

   $$current\ media\ buffer\ depth > (FEC_L * FEC_D * 2) + Packet\ Margin$$

   Packet Margin is set to 16 for FEC recovery process latency.
4. Current media buffer depth is updated after the requested media packet is sent out of the core.

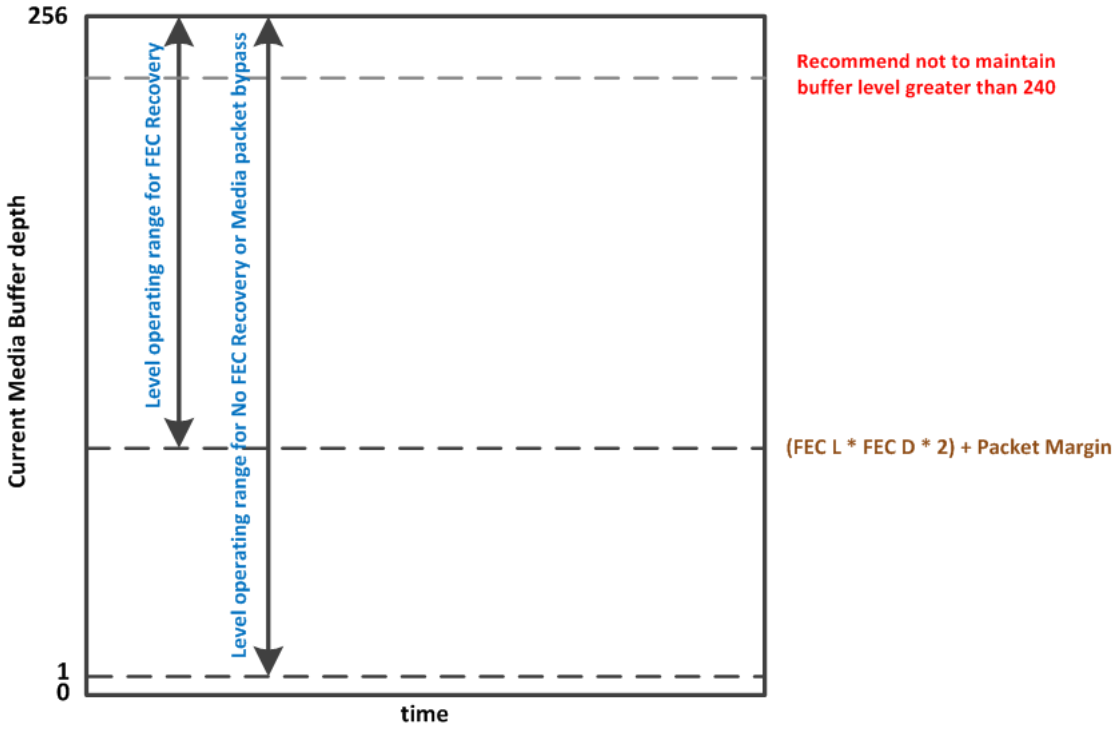


Figure 3-8: **Current Media Buffer Depth Maintained by the Controller**

Clocking

The VoIP FEC Receiver core has two clock domains:

- Core clock domain, `core_clk`.
  All the AXI4-stream and AXI4 interfaces, along with the main core activities are under this clock. (Fmin for `core_clk` is arbitrarily set at 100 Mhz. This is based on a calculated assumption that the core is able to handle an input bandwidth of 1G at this frequency.)
- AXI4-Lite clock domain, `s_axi_aclk`.
  Core register access works with this clock. (Fmin for `s_axi_aclk` is set at 50 MHz.)

Resets

The VoIP FEC Receiver core has 2 resets:

- Core domain reset, `core_reset`.
- AXI4-Lite domain reset, `s_axi_aresetn`

The resets must be synchronous to their individual clock domains. A minimum of 16 clock cycles is recommended for the reset assertion. Reset signal `s_axi_aresetn` is to be de-asserted last.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 3]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP Integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 3] for detailed information. IP Integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

Vivado Integrated Design Environment

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6].

Note: Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

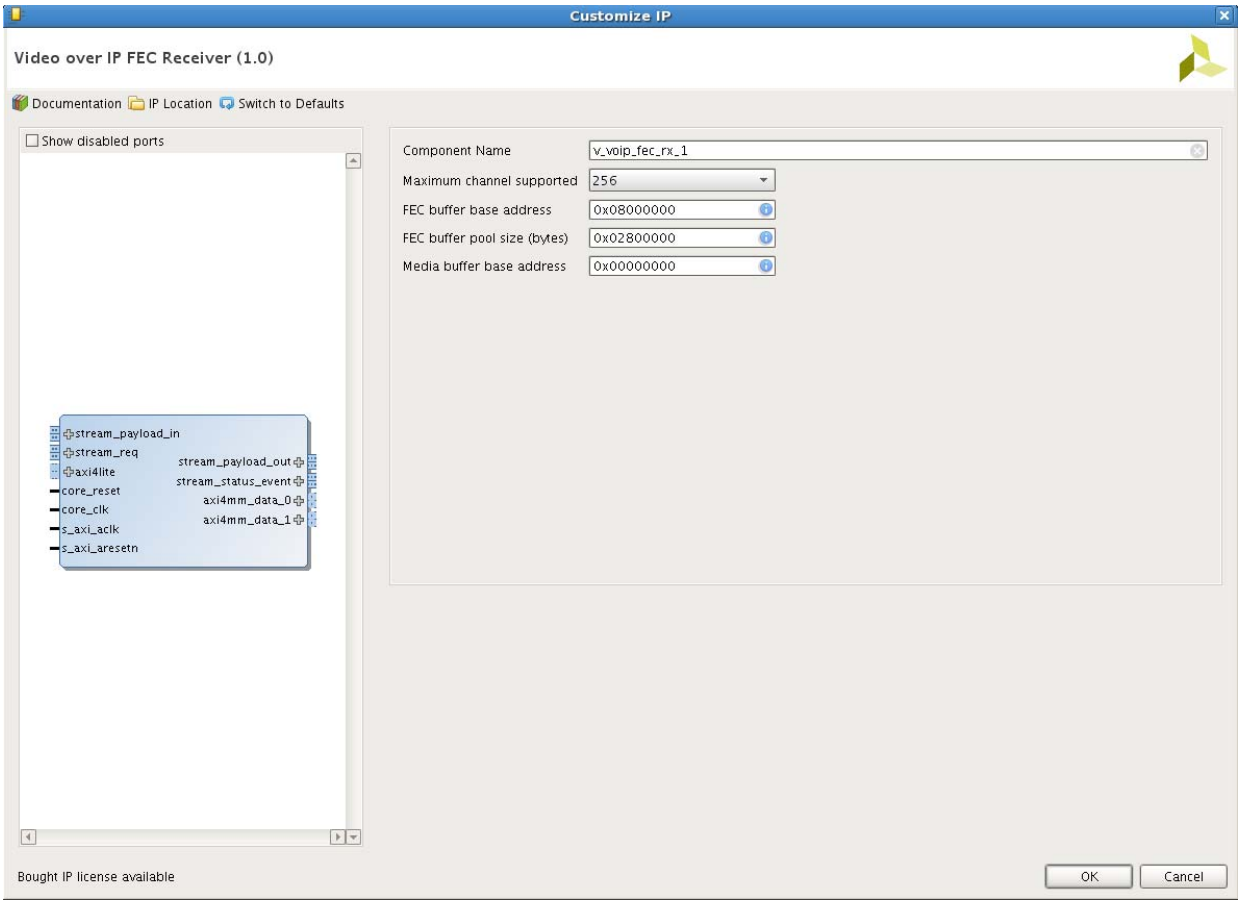


Figure 4-1: **Video over IP FEC Receiver GUI**

The Vivado IDE displays a representation of the IP symbol on the left side and the parameter assignments on the right, as follows:

- **Component Name**: The base name of output files generated for the module. Names must begin with a letter and must be composed of characters a to z, 0 to 9 and "_". The name v_voip_fec_rx_v1_0 cannot be used as a component name.
- **Maximum channel supported:** Specifies the number of channels (Note: Changing Maximum channel supported requires proper re-setting of Media buffer base address, FEC buffer base address and FEC buffer pool size).
- **Media buffer base address:** Specifies base address for media payload buffer (Note: Changing Media buffer base address requires proper re-setting of FEC buffer base address).
- **FEC buffer base address:** Specifies FEC buffer base address for "Maximum channel supported" checked (Note: Changing FEC buffer base address requires proper re-setting of Media buffer base address).
- **FEC buffer pool size (bytes):** Specifies FEC buffer pool size in terms of byte for “Maximum channel supported” checked.

Configuring the GUI Parameters

Table 4-1 describes the GUI parameter setting.

Table 4-1: **Variables for GUI Parameters Setting**

Parameters	Variables
$Supported\ Channel$	256/512
$Packet\ Size_{Bytes}$ (Bytes)	2048
$Media\ Buffer\ Depth_{channel}$ (packets)	256
$(FEC\ L + FEC\ D)_{max}$ (packets)	25

Setting the FEC Buffer Pool Size (Bytes)

$$FEC\ Packet\ Buffered_{channel} = ((FEC\ L + FEC\ D)_{max} * 2) + Packet\ Margin$$

Note: Packet Margin is set to 30 Packets in GUI default to give extra storage

$$\boldsymbol{FEC\ Buffer\ Pool\ Size\ (Bytes)} = Supported\ Channel * Packet\ Size_{Bytes} * FEC\ Packet\ Buffered_{channel}$$

Setting the FEC Buffer Base Address

$$Packet\ Buffer\ Size_{supp.\ channels} = Supported\ Channel * Packet\ Size_{Bytes} * Media\ Buffer\ Depth_{channel}$$

While setting the **FEC buffer base address**, should satisfy either of below rules:

1. $\boldsymbol{FEC\ buffer\ base\ address} \leq Media\ Buffer\ Base\ Address - FEC\ Buffer\ Pool\ Size$
2. $\boldsymbol{FEC\ buffer\ base\ address} \geq Media\ Buffer\ Base\ Address + Packet\ Buffer\ Size_{supp.\ channels}$

Use Case

Table 4-2 shows the examples value while configuring Media buffer base address, FEC buffer base address and FEC buffer pool size which based from the formula shown in Configuring the GUI Parameters.

Table 4-2: **Media Buffer Base Address, FEC Buffer Base Address, and FEC Buffer Pool Size Use Case**

Maximum channel supported	Media buffer base address	FEC buffer base address	FEC buffer pool size (bytes)
256	0x0000_0000	0x0800_0000	0x0280_0000
512	0x0000_0000	0x1000_0000	0x0500_0000

Note: Assuming Memory Base Address starts at 0x0000_0000

User Parameters

Table 4-3 shows the relationship between the GUI fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl console).

Table 4-3: **GUI Parameter to User Parameter Relationship**

GUI Parameter/Value[(1)(2)]	User Parameter/Value[(1)]	Default Value
Maximum channel supported	C_NUM_OF_CHANNELS	256
FEC buffer base address	C_FEC_BASE_ADDR	0x08000000
FEC buffer pool size (bytes)	C_FEC_POOL_SIZE	0x02800000
Media buffer base address	C_MEDIA_BASE_ADDR	0x00000000

1. Parameter values are listed in the table where the GUI parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.
2. The FEC and Media buffer base addresses should not overlap.

Output Generation

The Vivado design tools generate the files necessary to build the core and place those files in the `<project>/<project>.srcs/sources_1/ip/<core>` directory. For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

Constraints required for the core are clock frequency constraints for the clock domains described in Clocking in Chapter 3. Paths between the clock domains are constrained with a `max_delay` constraint and use the data path only flag, causing setup and hold checks to be ignored for signals that cross clock domains. These constraints are provided in the XDC constraints file included with the core.

Device, Package, and Speed Grade Selections

There are no device, package or speed grade requirements for this core. This core has not been characterized for use in low-power devices.

Clock Frequencies

See Maximum Frequencies in Chapter 2.

Clock Management

There are no clock management constraints for this core.

Clock Placement

There are no clock placement constraints.

Banking

There is no specific banking rule for this core.

Transceiver Placement

There is no transceiver placement constraints for this core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for this core.

Simulation

This section contains information about simulating IP in the Vivado Design Suite. For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Synthesis and Implementation

This section contains information about synthesis and implementation in the Vivado Design Suite. For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

As shown in Figure 5-1, the demonstration test bench is a simple System Verilog module which configures and tests the VoIP Forward Error Correction Receiver core. The test bench consists of several modules like RTP packet generator for generating and driving RTP packets over transport stream, APIs and Drivers for configuring the core, Checker for integrity check of packet coming out of core. The test bench is supplied as part of the Example Simulation output product group.

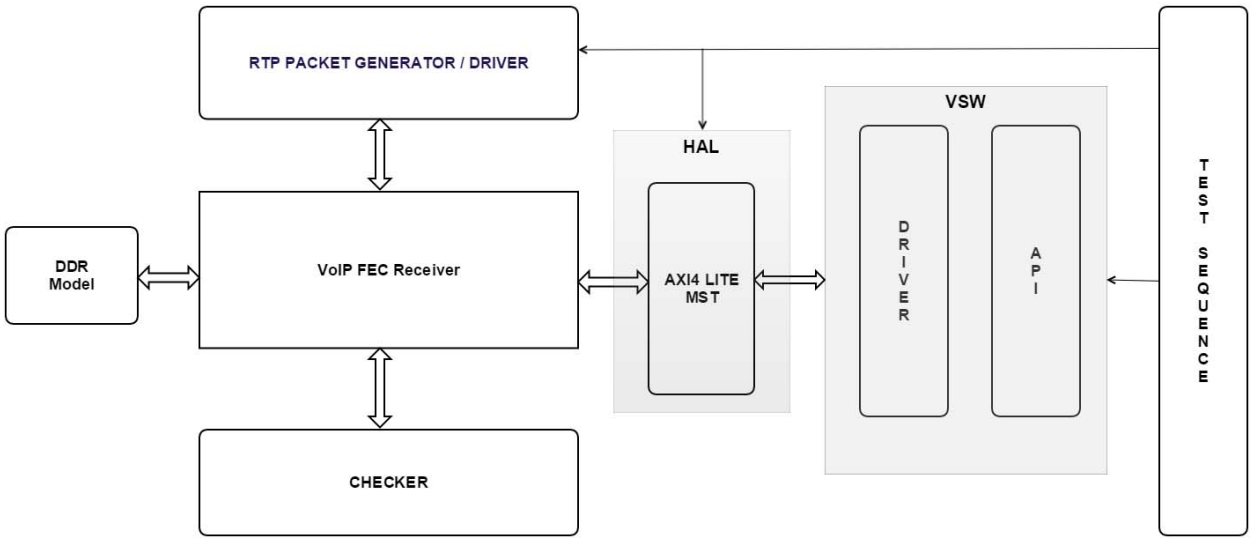


Figure 5-1: **Video over IP FEC Receiver Test Bench**

The main components of Demonstration test bench are described below:

RTP Packet Generaton/Driver: This module generates Real Time protocol packet over Transport stream and drives it to VoIP Forward Error correction Receiver core across all the enabled channels.

Checker: Packet checker module receives the packet from VoIP FEC receiver core and does packet pay load data integrity check, channel number mismatch check and packet size check

HAL: Hardware Access Layer is the register configuration layer. This layer has register read and write process.

VSW: Virtual Software layer. This layer consists of Driver and API. They control the Core configuration and are driven to Core by HAL. This layer is controlled using test case.

DDR model: This is Dummy DDR model used to store the RTP and FEC packets from core.

XILINX
ALL PROGRAMMABLE™

Appendix A

Verification, Compliance, and Interoperability

The Video over IP FEC Receiver core has been validated using the Xilinx® Kintex®-7 FPGA Broadcast Connectivity Kit.

Migrating and Upgrading

This appendix contains information about migrating a design from the ISE® Design Suite to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Parameter Changes

There are no parameter changes.

Port Changes

There are no port changes.

Other Changes

There are no other changes.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

TIP: *If the IP generation halts with an error, there might be a license issue. See License Checkers in Chapter 1 for more details.*

Finding Help on Xilinx.com

To help in the design and debug process when using the core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

The Solution Center specific to the Video over IP FEC Receiver core is listed below.

- Xilinx Ethernet IP Solution Center
- Xilinx MIG Solution Center
- Xilinx Solution Center for PCI Express

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the Video over IP FEC Receiver

AR: 54548

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

There are many tools available to address Video over IP FEC Receiver design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 9].

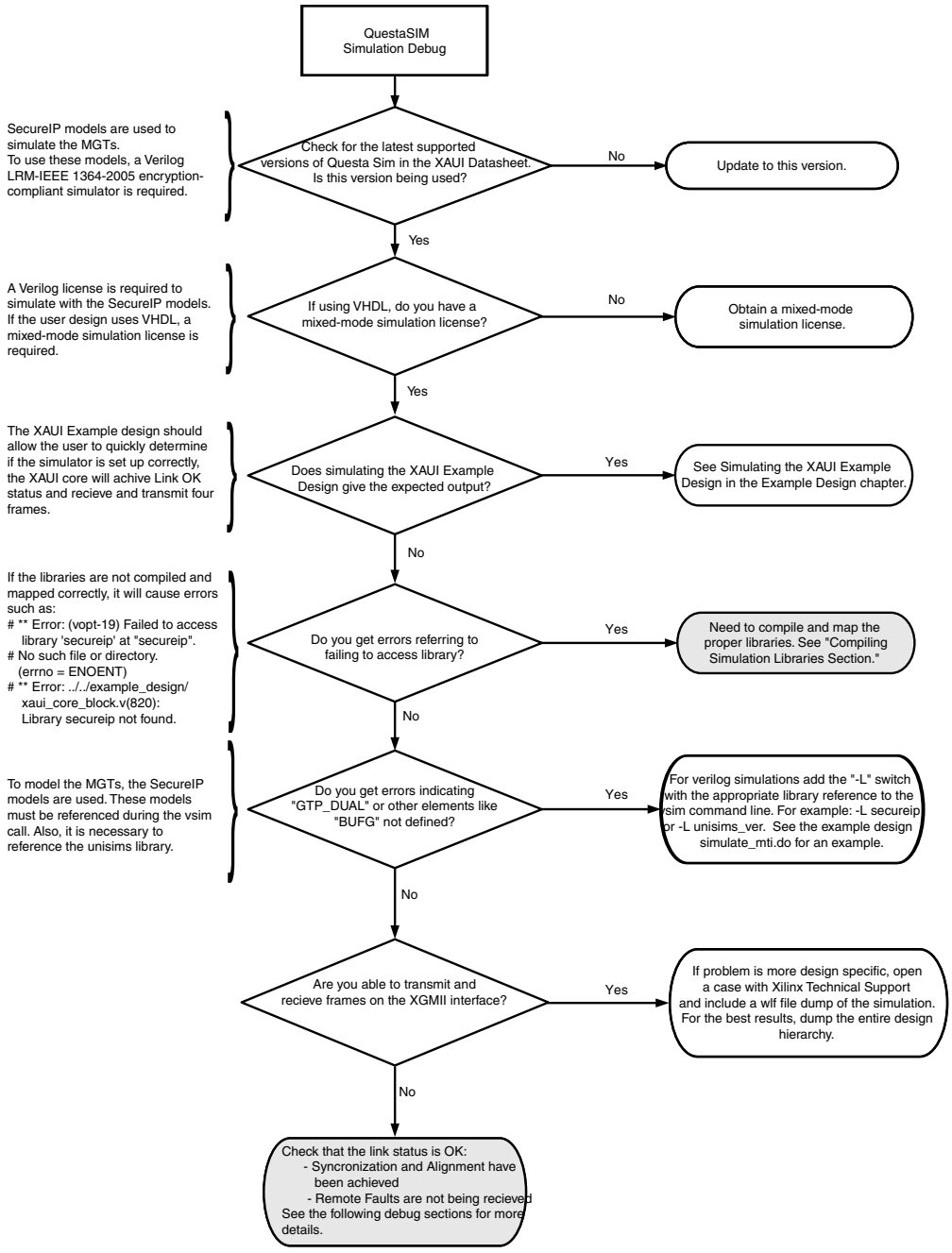
Reference Boards

Various Xilinx development boards support the Video over IP FEC Receiver. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series FPGA evaluation board KC705

Simulation Debug

The simulation debug flow for Questa® SIM is illustrated in Figure C-1. A similar approach can be used with other simulators.



Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado lab tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado lab tools for debugging the specific problems.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `locked` port.
- If your outputs go to 0, check your licensing.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk` and `aclk` inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.
- If the simulation has been run, verify in simulation and/or a Vivado Lab tools capture that the waveform is correct for accessing the AXI4-Lite interface.

AXI4-Stream Interfaces

If data is not being transmitted or received, check the following conditions:

- If transmit `<interface_name>_tready` is stuck Low following the `<interface_name>_tvalid` input being asserted, the core cannot send data.
- If the receive `<interface_name>_tvalid` is stuck Low, the core is not receiving data.
- Check that the `aclk` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed.
- Check core configuration.

Other Interfaces

IP Core Debug

Crucial Core and Register Setting

When generating core in Vivado, ensure that the base address for FEC Buffer Base Address, FEC buffer pool Size & Media Buffer Base Address is set based on the recommended computations described in Memory Bandwidth Consumption in Chapter 3.

Debug Operation

1. If a non-zero value is observed while reading valid_pkts_cnt (0x090), it indicates the core is receiving valid packets.

2. If a non-zero value is observed while reading oob_pkts_cnt(0x0A4), it indicates that the media buffer has overflowed, output has gone out of sync and a channel recovery need to be performed.

3. Stream changes on a particular channel (disruption in the source packet sequence number) would require a channel recovery to be performed.

4. Software configuration change on a particular channel (reconfiguring chan_conf (0x080) on the fly) would require a channel recovery to be performed.

Refer to Figure C-2.

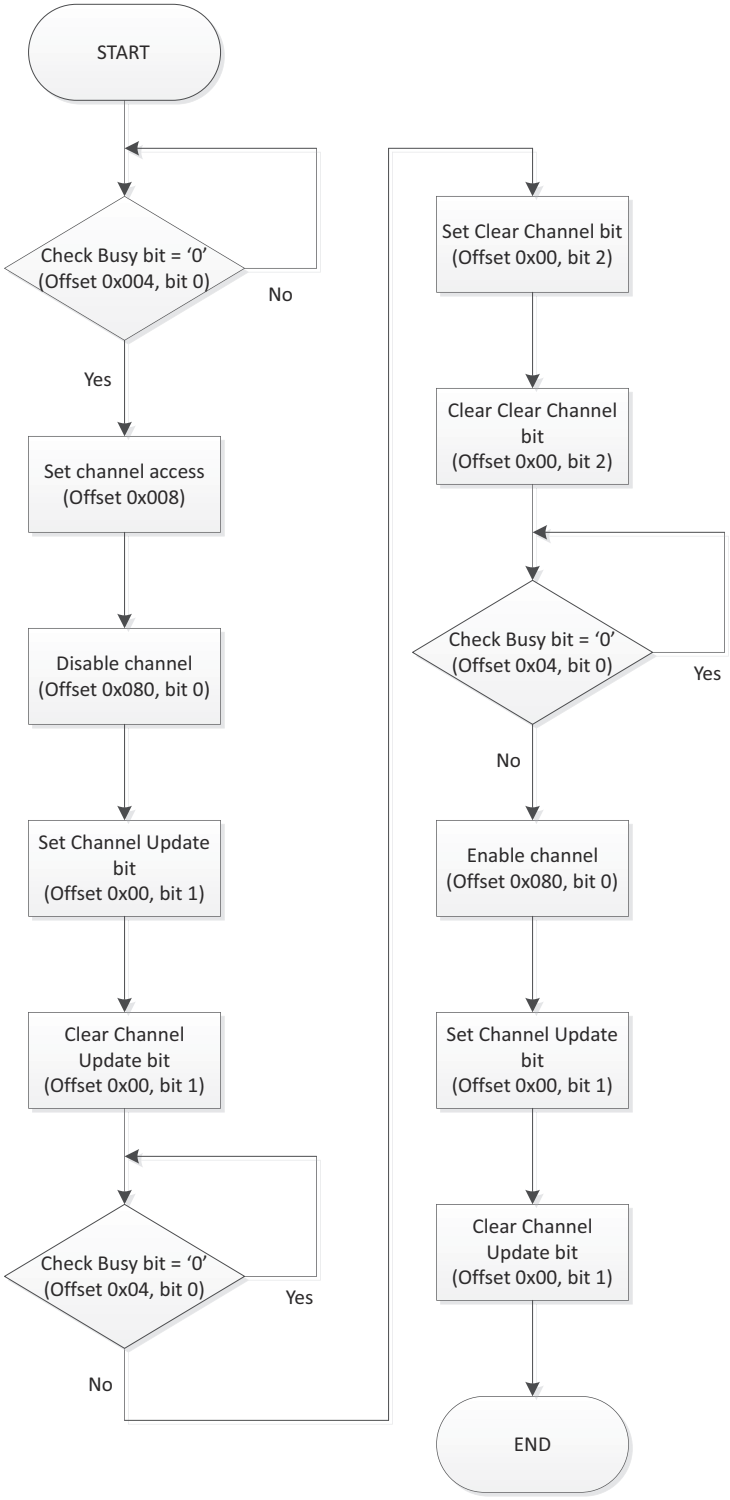


Figure C-2: **Channel Recovery Flowchart**

Additional Resources and Legal Notices and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. SMPTE 2022-5,6 Video Over IP Product Page
2. SMPTE 2022-1,2 Video Over IP *Product Page*
3. *Vivado® Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
4. *Vivado Design Suite User Guide: Designing with IP* (UG896)
5. *Vivado AXI Reference Guide* (UG1037)
6. *Vivado Design Suite User Guide: Getting Started* (UG910)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *ISE® to Vivado Design Suite Migration Guide* (UG911)
9. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
10. *Vivado Design Suite User Guide - Implementation* (UG904)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	1.0	Added support for UltraScale+ families. Added a new register, oor_pkts_cnt (0x0B0).
04/01/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices