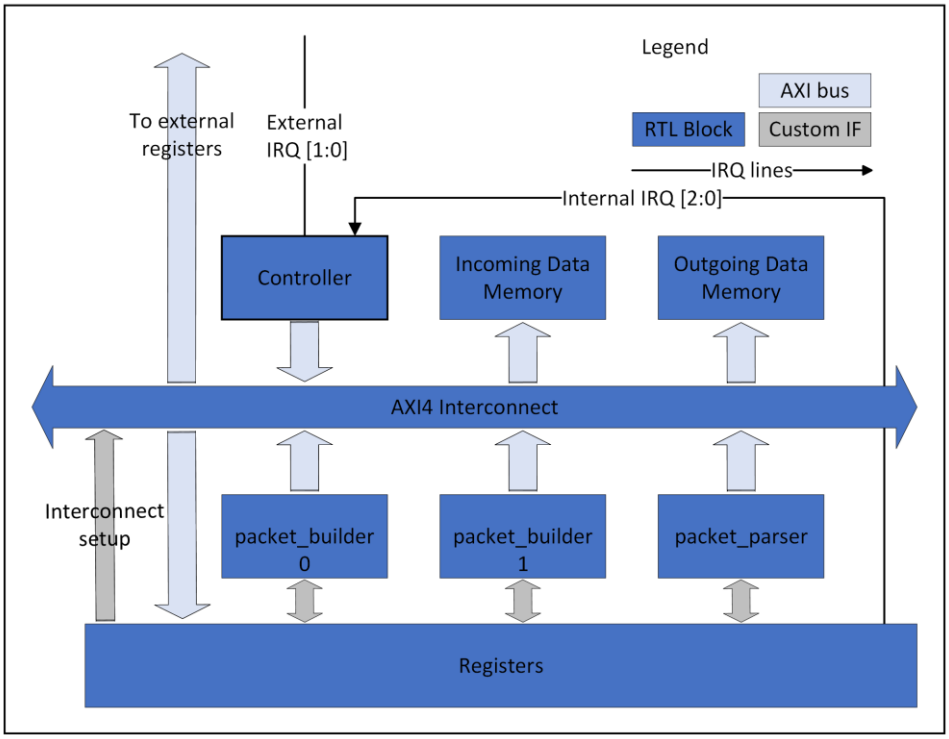
1. System overview

The described system is an AXI4-based system operating in a single clock domain. The main purpose of the system is packet processing according to given transaction level protocol. It should prepare packets from incoming raw data and parse incoming packets to extract packet info and possible transmission errors. The system consists of following blocks:

- Controller with external and internal IRQ lines
- AXI4 interconnect with configurable priority
- Incoming data memory (64kB)
- Outgoing data memory (64kB)
- Registers block
- Two packet_builder (PB) blocks
- Single packet_parser (PP) block
- (External registers block)



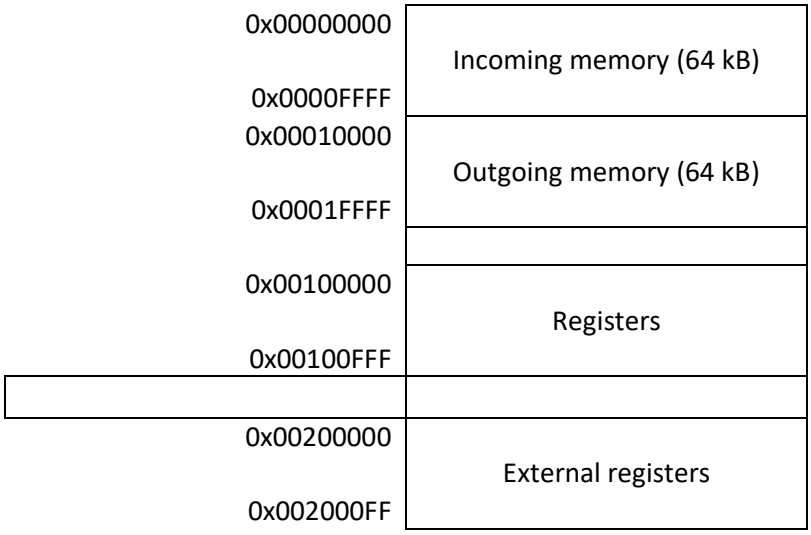
1.1. Operating scenarios

There are two main scenarios of system behavior. The first one is a packet building task, the second one is a packet parsing task. The system has two external interrupt lines each for each task. Number of the line determines the task. Line #0 (ext_irq[0]) indicates that the data for packet building task is ready in the incoming memory, line #1 (ext_irq[1]) – data for parsing task. As there are two packet_builder blocks (PB0 and PB1), two consecutive interrupts from line #0 can be handled. If both packet_builder blocks are available, controller chooses one with the smallest number (PB0). Only one packet parsing task can be running at the same time. If the requested task can't be accomplished (both PB are busy in case of ext_irq[0] or PP is busy in case of ext_irq[1]), the controller should indicate to external registers and clear the interrupt without waiting for chosen block to finish its task.

Packet building task sequence: after receiving an external interrupt #0 (ext_irq[0]) the controller reads packet configuration from external registers, sets up the register block according to the given packet_builder block and clears the interrupt. The configuration of the block includes start address of data in incoming memory, start address in outgoing memory, count of bytes, packet header value, ECC and CRC calculation options and data merging option. After the register setup controller enables the block, it fetches the register values and starts the task. It reads data from incoming memory and builds packet by adding header, merging data and calculating ECC for the header and CRC for data based on the fetched register values. Then it writes built packet to given address in the outgoing memory. After writing the packet it raises the interrupt to the controller. Controller clears the interrupt. The task finishes.

Packet parsing task sequence: after receiving an external interrupt #1 (ext_irq[1]) the controller reads packet configuration from external registers, sets up the registers related to the packet_parser (PP) block and clears the interrupt. The configuration of the block includes packet header address in memory and ECC/CRC errors ignoring. After the register setup controller enables the block, it fetches the register values and starts the task. Block reads the header and calculates its ECC. If ECC has an uncorrectable error and ECC ignoring is not set, block finishes the task by writing the packet information to register and raises interrupt. If there are no uncorrectable ECC errors or they are ignored, block starts reading packet data from incoming memory and calculating its CRC. After all data has been read it compares calculated CRC with CRC written with the packet in incoming memory, writes packet information to register and raises interrupt. Controller clears the interrupt. The task finishes.

1.2. Address map



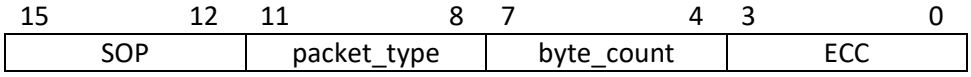
1.3. Packet structure

Each packet in the system consists of 3 fields: 2 bytes for header, 1-16 bytes for data and 1 byte for CRC8.

Field	Header	Data	CRC8
Length, bytes	2	1-16	1

The header has 4 fields

- SOP – start of packet marker, default value = 0xA;
- packet_type – type of packet, can take any value;
- byte_count – count of bytes in packet. As packet should have at least 1 byte of data, actual count of bytes = byte_count + 1;
- ECC – Hamming error correction code 12 8, calculated from bits [11:4] of header. Can detect 2-bit error and correct single-bit error.



2. Blocks description

2.1. Controller

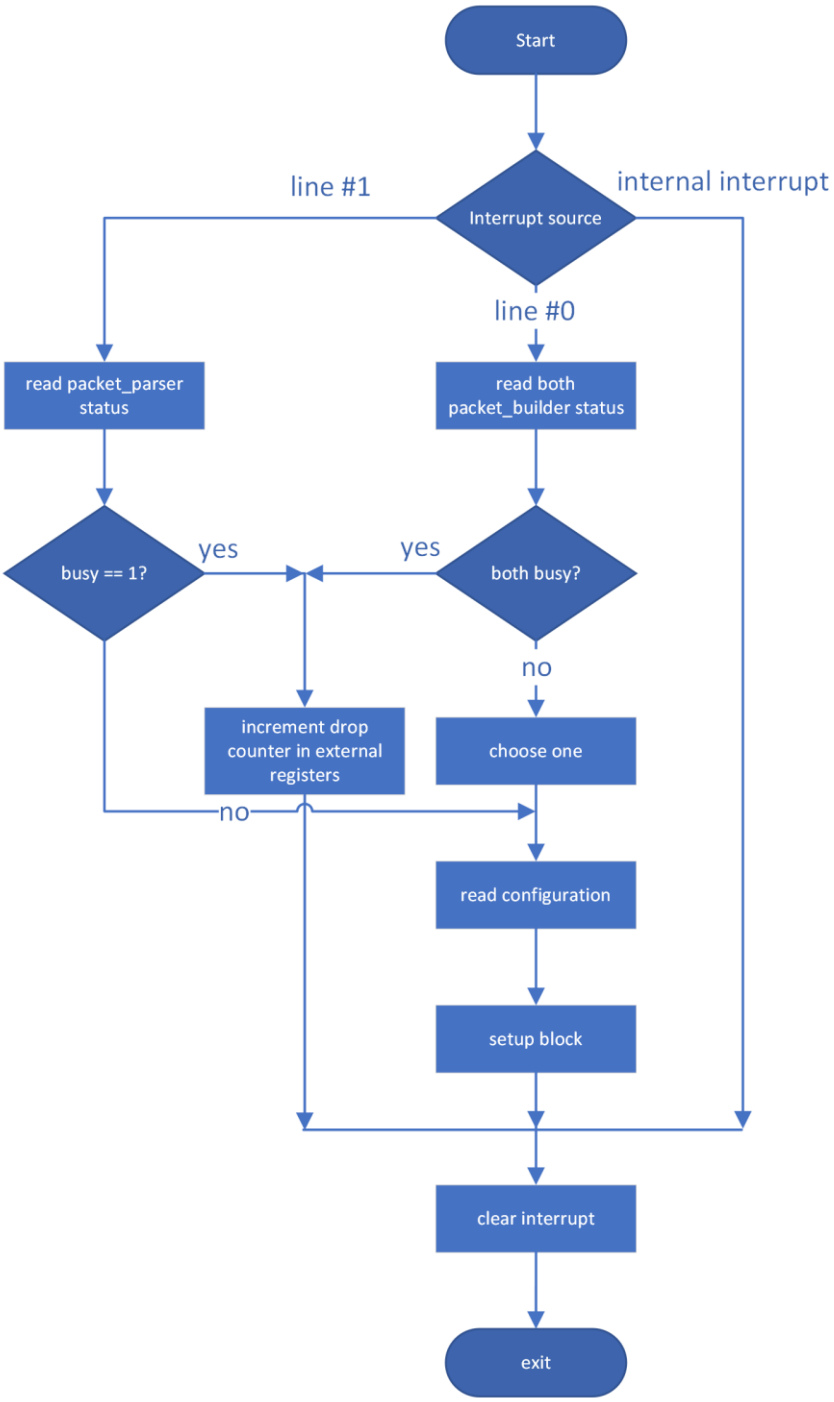
The controller serves to set up the packet processing blocks using configuration written to the external registers. Most of the time the controller is idle waiting for incoming interrupts.

2.1.1. Interface description

The controller has a single master AXI Interface and 5 incoming interrupt lines. Two of the interrupt lines are external interrupts (ext_irq[1:0]), which indicate that the data is ready at the incoming memory and block configuration is valid in the external registers. Three interrupt lines are internal interrupts (int_irq[2:0]). All interrupts are edge triggered. As the controller doesn't operate large amounts of data, it would be enough to have AXI4-Lite functionality, but the interface to the interconnect should match the AXI4 Interface.

2.1.2. Interrupt handling

After receiving an interrupt, the controller enters the interrupt handling routine. If the interrupt is an internal interrupt, the controller should clear the interrupt bit in the corresponding register. If the interrupt is external, the controller should check whether the task requested by the interrupt can be fulfilled. For example, if it's a PB task, the controller should check PB0 and PB1 busy flags. If both blocks are busy, the controller increments dropped tasks counter in an external register and clears the interrupt. If the task can be fulfilled, the controller reads block configuration from external registers, writes it to the corresponding block's registers, launches the block and clears the interrupt.



2.2. Registers block

Registers block consists of several 32b registers. It should support byte access. It shouldn't support bursts unless the controller supports bursts.
The block has three interrupt lines leading to the controller. It should rise an interrupt when one of the packet processing blocks writes 1 to corresponding register through internal interface and hold it for at least 1 clock cycle.

Correspondence of internal interrupt lines to registers block interrupts:
int_irq[0] – pb0_irq
int_irq[1] – pb1_irq

int_irq[3] – pp_irq

Following table contains external registers description. Bits not included to the bitmap are reserved, they should read as 0, write shouldn't affect them. Abbreviations for register types are revealed below:
RW – read-write
RO – read-only
W1C – write '1' to clear

Address	Name	Type	Bitmap	Field	Reset value	Description
0x0	SYS_CFG	RW	[0]	priority_sel	0	AXI Interconnect arbitration: 0: Round-Robin 1: priority based, priority_num defines the master with the highest priority
			[2:1]	priority_num	0	Number of master with highest priority for AXI Interconnect
0x4	PB0_STS	RO	[0]	pb0_busy	0	Busy status of PB0 0: idle 1: busy
0x8	PB0_CTRL0	RW	[0]	pb0_start	0	PB0 start signal. (it should generate pulse upon writing 1)
0xC	PB0_CTRL1	W1C	[0]	pb0_irq	0	PB0 interrupt flag. Write 1 to clear
0x10	PB0_CTRL2	RW	[31:0]	pb0_in_addr	0x0	Start address of data in memory. **Should be aligned to word boundary (pb0_in_addr [1:0] = 0)**
0x14	PB0_CTRL3	RW	[3:0]	pb0_byte_cnt	0x0	Number of bytes for building a packet
			[7:4]	pb0_pkt_type	0x0	Value for packet_type field
			[8]	pb0_ecc_en	0x0	0: don't calculate ECC, use predefined value 1: calculate ECC
			[9]	pb0_crc_en	0x0	0: don't calculate CRC, use predefined value 1: calculate CRC
			[11:10]	pb0_ins_ecc_err	0x0	0: don't insert ECC errors 1: insert single ECC error

						2, 3: insert double ECC error
			[12]	pb0_ins_crc_err	0x0	0: don't insert CRC error 1: insert CRC error
			[16:13]	pb0_ecc_val	0x0	ECC value to use if ecc_enable = 0
			[23:17]	pb0_crc_val	0x0	CRC value to use if crc_enable = 0
			[27:24]	pb0_sop_val	0xA	SOP field value
			[31:28]	pb0_data_sel	0x0	Defines data merging options 0: use the least significant byte from every word 1: use only first 2 bytes from every word 2: to be continued
0x18	PB0_CTRL4	RW	[31:0]	pb0_out_addr	0x0	Start address in the outgoing memory. **Should be aligned to word boundary (pb0_out_addr[1:0] = 0)**
0x1C	PB1_STS	RO	[0]	pb1_busy	0	Busy status of PB1 0: idle 1: busy
0x20	PB1_CTRL0	RW	[0]	pb1_start	0	PB1 start signal. (should generate pulse upon writing 1)
0x24	PB1_CTRL1	W1C	[0]	pb1_irq	0	PB1 interrupt flag. Write 1 to clear
0x28	PB1_CTRL2	RW	[31:0]	pb1_in_addr	0x0	Start address of data in memory. **Should be aligned to word boundary (pb1_in_addr[1:0] = 0)**
0x2C	PB1_CTRL3	RW	[3:0]	pb1_byte_cnt	0x0	Number of bytes for building a packet
			[7:4]	pb1_pkt_type	0x0	Value for packet_type field
			[8]	pb1_ecc_en	0x0	0: don't calculate ECC, use predefined value 1: calculate ECC
			[9]	pb1_crc_en	0x0	0: don't calculate CRC, use predefined value 1: calcucate CRC
			[11:10]	pb1_ins_ecc_err	0x0	0: don't insert ECC errors 1: insert single ECC error 2, 3: insert double ECC error
			[12]	pb1_ins_crc_err	0x0	0: don't insert CRC error

						1: insert CRC error
			[16:13]	pb1_ecc_val	0x0	ECC value to use if ecc_enable = 0
			[23:17]	pb1_crc_val	0x0	CRC value to use if crc_enable = 0
			[27:24]	pb1_sop_val	0xA	SOP field value
			[31:28]	pb1_data_sel	0x0	Defines data merging options 0: use the least significant byte from every word 1: use only first 2 bytes from every word 2: to be continued
0x30	PB1_CTRL4	RW	[31:0]	pb1_out_addr	0x0	Start address of data in memory. **Should be aligned to word boundary (pb1_out_addr[1:0] = 0)**
0x34	PP_STS	RO	[0]	pp_busy	0	Busy status of PP 0: idle 1: busy
			[1]	pp_pkt_ ecc_corr	0x0	ECC correctable error flag
			[2]	pp_pkt_ecc_uncorr	0x0	ECC uncorrectable error flag
			[3]	pp_pkt_crc_err	0x0	CRC error flag
			[7:4]	pp_pkt_byte_cnt	0x0	Packet byte count field
			[11:8]	pp_pkt_type		Packet type field
0x38	PP_CTRL0	RW	[0]	pp_start	0	PP start signal. (should generate a pulse upon writing 1)
0x3C	PP_CTRL1	W1C	[0]	pp_irq	0	PP interrupt flag. Write 1 to clear
0x40	PP_CTRL2	RW	[31:0]	pp_header_addr	0x0	Address of packet header in incoming memory
0x44	PP_CTRL3	RW	[0]	pp_ignore_ecc_err	0x0	0: read data and calculate crc even if there is ecc uncorrectable error 1: if ecc is corrupted and uncorrectable, don't read data for crc calculating

2.3. External registers

External registers are accessed only by the controller. The block consists of several 32b registers and stores packet processing block configuration and external interrupt flags. It should support byte access. It shouldn't support bursts unless the controller supports bursts.

Following table contains external registers description. Bit fields description matches bit fields description from registers block table.

Address	Name	Type	Bitmap	Field
0x0	EXT_PB_CTRL1	W1C	[0]	irq_line_0
0x4	EXT_PB_CTRL2	RO	[31:0]	in_addr
0x8	EXT_PB_CTRL3	RO	[3:0]	byte_cnt
			[7:4]	packet_type
			[8]	ecc_enable
			[9]	crc_enable
			[11:10]	insert_ecc_err
			[12]	insert_crc_err
			[16:13]	ecc_val
			[23:17]	crc_val
			[27:24]	SOP_val
			[31:28]	data_sel
0xC	EXT_PB_CTRL4	RO	[31:0]	out_addr
0x10	EXT_PP_CTRL1	W1C	[0]	irq_line_1
0x14	EXT_PP_CTRL2	RO	[31:0]	header_addr
0x18	EXT_PP_CTRL3	RO	[0]	ignore_ecc_err
0x1C	EXT_DROP_CNT	RW	[31:0]	drop_cnt*

*Field drop_cnt is used by the controller to indicate how many tasks were dropped while the system was busy.

2.4. AXI Interconnect

The AXI Interconnect serves to connect different masters with different slaves. It has 4 Master ports (Controller, PB0, PB1, PP) and 4 Slave ports (Registers, Incoming memory, Outgoing memory, External Registers). The interconnect should support bursts and different arbitration rules: Round-Robin and with given priority. Arbitration rules and priority should be set up using the registers block.

2.5. Incoming memory block

Incoming memory consists of 16384 32bits cells. It should support read and write bursts as well as byte access.

2.6. Outgoing memory block

Outgoing memory consists of 16384 32bits cells. It should support read and write bursts as well as byte access.

2.7. Packet building block (packet_builder)

The packet building block should start fulfilling the task after receiving the start pulse from the register interface (if the Registers block doesn't support pulse generation, the block must extract it itself). Then the block sets busy signal, fetches configuration provided by the register interface and builds the packet by reading data from incoming memory, adding header and calculating CRC (based on config). When the packet is built, the block writes it to the outgoing memory, deasserts busy flag and generates an interrupt pulse.
The start address in incoming memory and the address in outgoing memory are assumed to be word-aligned (bits [1:0] of address are 0) to avoid complicated bytes manipulations. AXI transactions could be (but mustn't) burst transactions.

Port name	Direction	Bus width
Clocking and reset		
clk_i	I	1
rst_n_i	I	1
AXI4 Interface		
m_axi_o	O	
m_axi_i	I	
Register Interface		
start_i	I	1
busy_o	O	1
irq_o	O	1
addr_in_i	I	32
byte_cnt_i	I	4
pkt_type_i	I	4
ecc_en_i	I	1
crc_en_i	I	1
ins_ecc_err_i	I	2
ins_crc_err_i	I	1
ecc_val_i	I	4
crc_val_i	I	8
sop_val_i	I	4
data_sel_i	I	4
addr_out_i	I	32

2.8. Packet parsing block (packet_parser)

The packet parsing block should start fulfilling the task after receiving start pulse from the register interface (if the Registers block doesn't support pulse generation, the block must extract it itself). Then the block sets busy signal, fetches configuration provided by the register interface and reads the packet header from incoming memory. Then it calculates the header ECC and compares it with the read ECC. If there is an ECC uncorrectable error detected, based on the configuration the block either reads the packet data from the incoming memory calculating it's CRC or indicates that an uncorrectable error was detected in the header and finishes the task.

When the packet parsing is done, the block updates its outputs, deasserts busy flag and raises an interrupt.

Port name	Direction	Bus width
Clocking and reset		
clk_i	I	1
rst_n_i	I	1
AXI4 Interface		
m_axi_o	O	
m_axi_i	I	
Register Interface		
start_i	I	1
busy_o	O	1
irq_o	O	1
addr_hdr_i	I	32
Ignore_ecc_err_i	I	1
pkt_ ecc_corr_o	O	1
pkt_ecc_uncorr_o	O	1
pkt_crc_err_o	O	1
pkt_byte_cnt_o	O	4
pkt_type_o	O	4