15400 Calhoun Drive, Suite 400
Rockville, Maryland, 20855
(301) 294-5200
http://www.i-a-i.com

Intelligent Automation Incorporated

CAGE100: Real-Time Multi-Port Packet Capture System for 100 Gigabit Ethernet Traffic

Final Technical Report

Reporting Period: Jun. 18, 2011 – Jun. 14, 2012

Contract No. DE-SC0006295
Sponsored by: DOE Office of Science
COTR/TPOC: Dr. Thomas Ndousse-Fetter

Prepared by
Shahin Farrokhnia
Ali Namazi
Babak Azimi-Sadjadi
Chujen Lin

1 Summary

In course of the PHASE I contract we designed a complete digital system for capturing a 10/40/100 Gigabit Ethernet frames (layer 1-4). A comprehensive design document prepared that explains the functional and logical specification of each design element.

The 40/100 Gigabit Ethernet physical coding sub-layer is identical to 10GBASE-R specification. Therefore for the PHASE I project where we had limited time, we developed a 10 Gigabit Ethernet capture system. This design is complied with 10GBASE-R spec and it has many commonalities with 40/100GBASE-R specs. Therefore in designing this system, we took a very modular design approach to systematically migrate it to the 40/100GBASE-R spec.

The system that we developed for 10 Gig Ethernet has a PHY and a Main Capture Engine board that are fully explained in the following sub-sections. The PHY board is responsible to deliver a replica of receiving and transmitting signals to the main capture engine while maintaining the connection between devices under test (DUTs). This PHY board has a 10 Gig fan-out buffer and two 10 Gig de-serializers. These high-speed chips were chosen from Vitesse Semiconductor, a leading supplier of IC solutions for high-speed network system.

Since 10-Gig is a fairly new technology, for the fan-out buffer chip, we could only find an engineering sample IC from Vitesse. At the time that we received the engineering samples and made the first version of PHY board, there were some known issues with the chip that were scheduled to be fixed in the next re-spin phase. Consequently, we were not able to fully test and verify the functionality of our 10-Gig Ethernet capture tool. However, we simulated and tested the logic design of capture engine, i.e. 10G capture engine module, DDR3 buffer memory, and 1G Ethernet host interface. It's worth mentioning that we are working with the manufacturer to troubleshoot the problem we encountered with the fan-out buffer chip.

1.1 FPGA Platform Selection and Design of PHY Board

Our very recent FPGA market study for solutions in 10 gigabit serial transceivers shows that FPGA families with 10.3125 Gbps multi-gigabit transceivers are not only very expensive (20k$) for a phase I project but also has a few months lead time that would not meet our deadline for the final project demo. Therefore for the Phase-I we decided to use a discrete 10Gig de-serializer and incorporate an affordable and yet high-end FPGA for processing. For future development of 40/100 gig capture tool we expect to have high speed embedded serial transceivers inside the FPGA.

1.1.1 Calculating the system clock rate inside FPGA

The serial transceivers for 10/40/100 Gig Ethernet technology deliver parallel signals with a speed of 156.25 MHz. This speed for each physical link (10/40/100 GHz) is derived from the following equations:

10GBASE-R : 10.3125 Ghz / 66-bit = 156.25 Mhz
40GBASE-R : 4 * 10.3125 Ghz / 4 * 66-bit = 156.25 Mhz
100GBASE-R : 10 * 10.3125 Ghz / 10* 66-bit = 156.25 Mhz

Therefore regardless of the physical link rate, the system clock speed is remained to a constant rate of 156.25 MHz. Of course, the complexity of digital processing of each links increase as the physical rates increase, i.e. for 10GBASE-R a single 66-bit data with speed of 156.25 MHz has to be processed whereas for 40GBASE-R 4 sets of 66-bit data each with 156.25 MHz is processed.

This feature of Ethernet protocol has two advantages. First that it makes the design very modular for migration to higher speeds and secondly a digital system with the speed of 156.25 MHz is very suitable to be implemented on FPGA platforms.

Based on this study, we purchased the Xilinx “ML605” Virtex-6 FPGA evaluation board from Xilinx and has a powerful V6LX240 FPGA with a 1GB DDR3 memory module for capture buffer, an Ethernet interface for Host Software interface, and a high-pin and high-speed connector to connect to the PHY board. This board is shown in Figure 1.

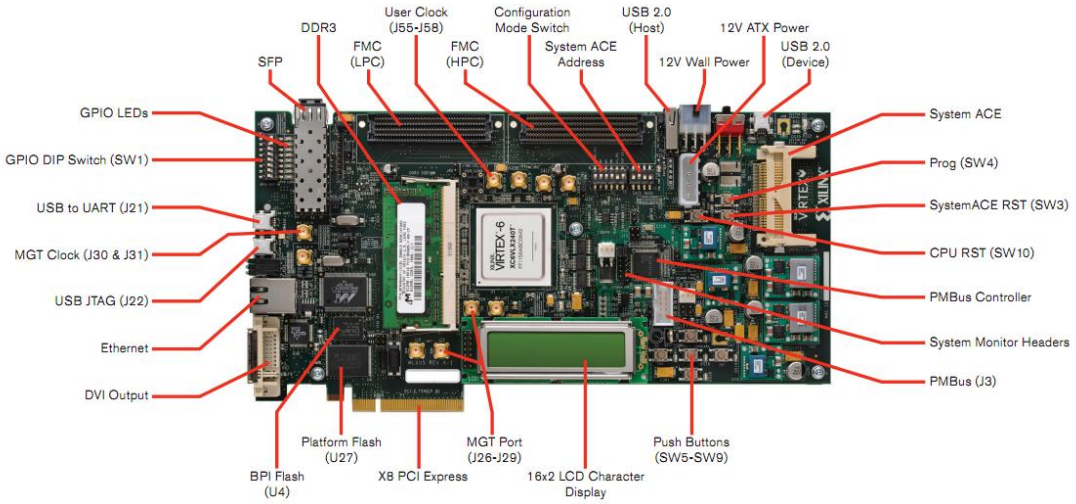

Figure 1. Xilinx ML605 Evaluation Board (picture is from Xilinx website)

We designed and added a PHY board to ML605 to receive a replica of signals from DUTs and provide a parallel signal to the FPGA. The PHY board has a 2-channel 10.3125 Gbps fan-out buffer (Vitesse) and two (RX, TX) 10.3125 Gbps de-serializers. The interface to the two devices under tests is two SFP+ optical connectors and the interface from PHY de-serializers to the FPGA board is a 400-pin high-speed connector. The PHY board block diagram is shown in Figure 2:

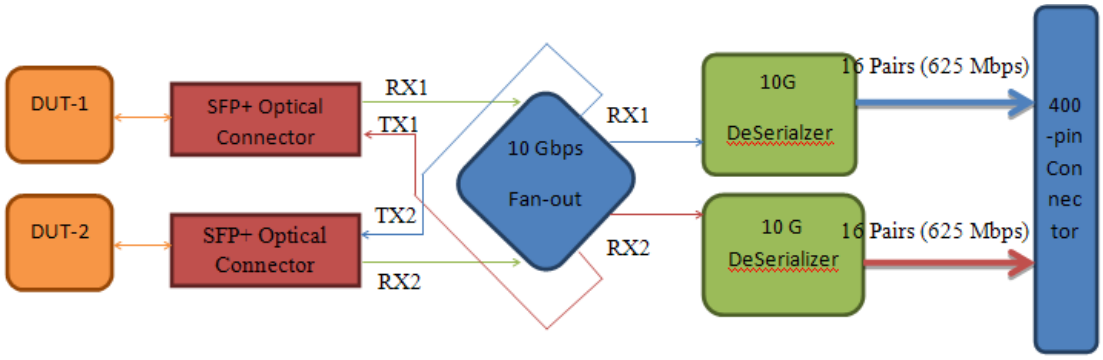

Figure 2 PHY board block diagram for 10GBASE-R

As shown in above block diagram the two devices under tests (DUT-1 & DUT-2) connect to the PHY board via two SFP+ optical connectors. SFP+ is a typical optical connector that supports rates up to 10 Gbps. The SFP+ connectors convert the optical signal into electrical signal RX and TX.

For 40/100GBASE-R links the similar approach can be used. In such cases a fan-out buffer with more channels (4 and 10 respectively) are utilized to deliver all the channels to the FPGA. Figure 3 and Figure 4 show the block diagram for 40GBASE-R and 100GBASE-R links.

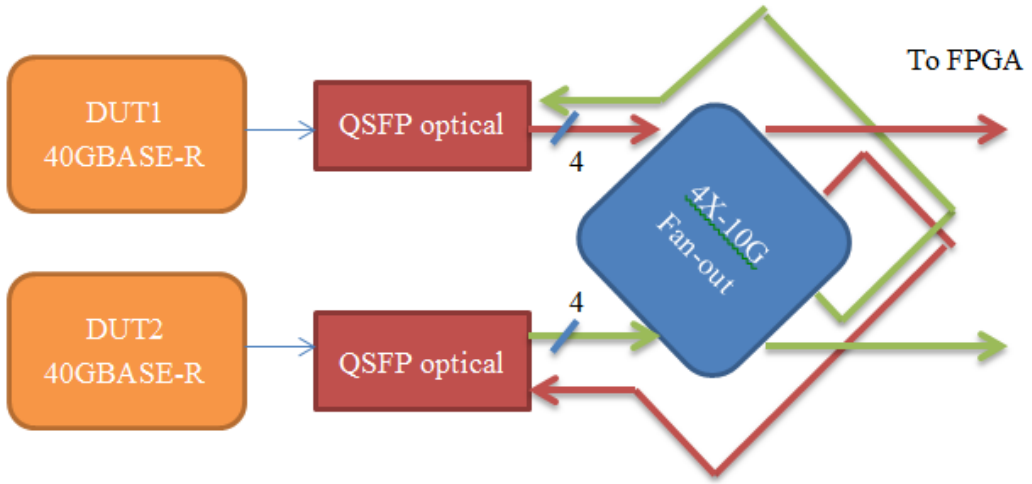

Figure 3 40GBASE-R PHY Board block diagram

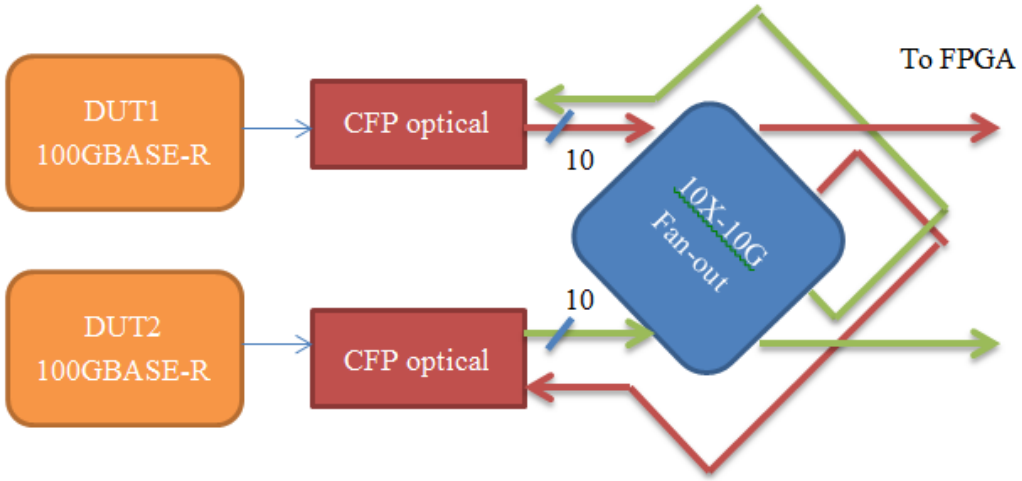

Figure 4 100GBASE-R PHY Board block diagram

The RX signals from each SFP+ goes to the 10 Gbps fan-out buffer. The fan-out buffer, VSC7111, is manufactured by VITESSE Semiconductors and can support rates up to 11.5 Gbps. The fan-out buffer makes two copies of each RX signals where one goes to the 10G De-Serializer and the other goes to the TX signal of other SFP+. In this way the 2 DUTs are connected and the capture processing has the least effect on the signal quality. The VSC7111 has both input signal equalization and output signal pre-emphasis circuitries that programmable with multiples settings. This feature is ideal for countering signal degradation over a wide variety of transmission media types and lengths.

There are two 10G de-serializers, VSC8479, manufactured by VITESSE Semiconductors that converts up to 10.3125 Gbps differential pair signal to a 16-pair

differential parallel signals with a rate of 625Mbps (10Gbps / 16). The two RX outputs from fan-out buffer goes to the two de-serializers and the parallel outputs are directed to the FPGA platform via a high-speed connector.

The PCB layout for the PHY board has been designed and manufactured in this performance period. The PHY lay out routed on an 18-layer board and extensive care has been taken to make sure that all signal integrity issues have been considered for 10 gig traces and other high-speed routes (32 pairs of 625 MHz LVDS signals). Figure 5 shows the PCB layout.

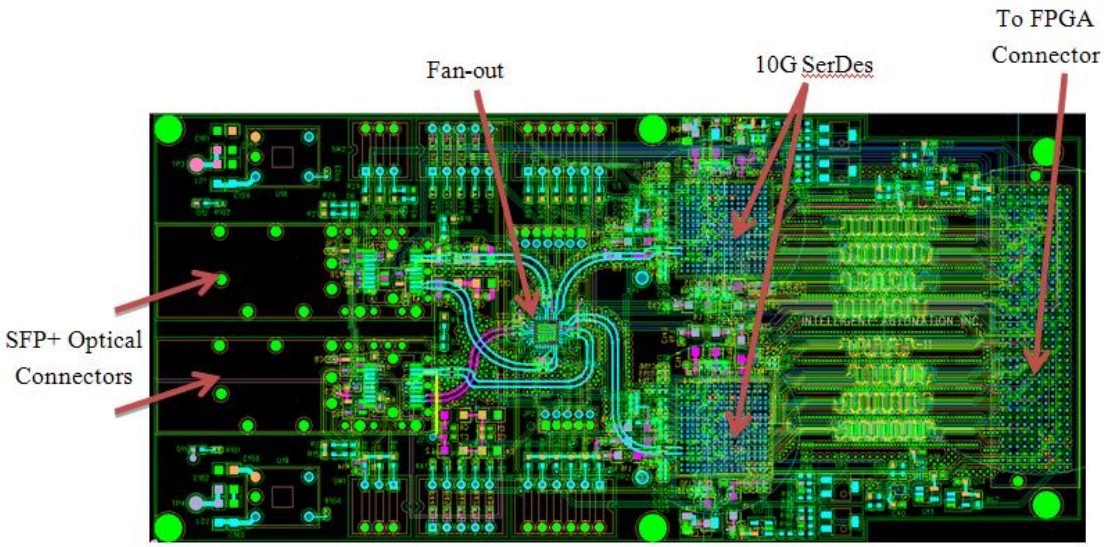

Figure 5 PHY PCB Layout

In order to test the system, we set up two computers with 10 Gig Ethernet cards as devices under tests. The two computers are connected with fiber cables via capture tool PHY board. In such test arrangment, the two computers communicate with each other and the capture tool can monitor the traffic. Figure 6 shows the test setup and Figure 7 shows the capture tool (Phy and Main boards).

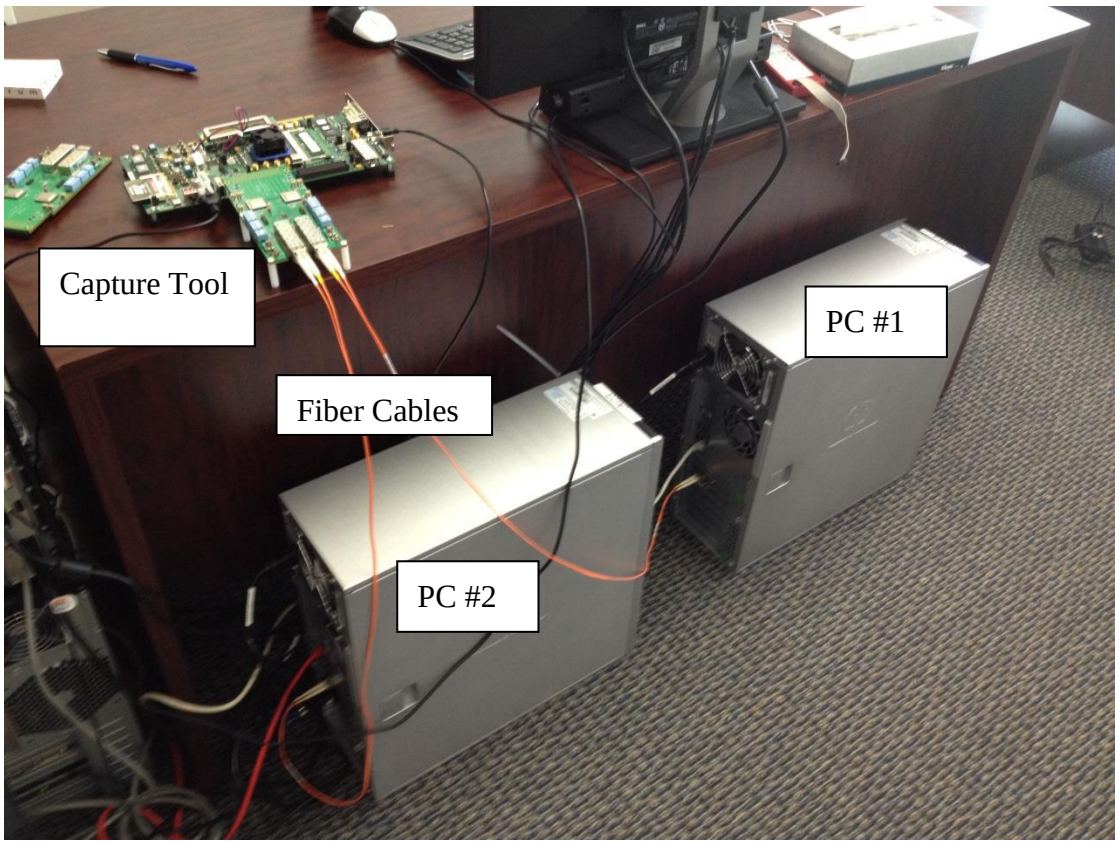

Figure 6 Test setup with 2 PCs connected via the capture tool

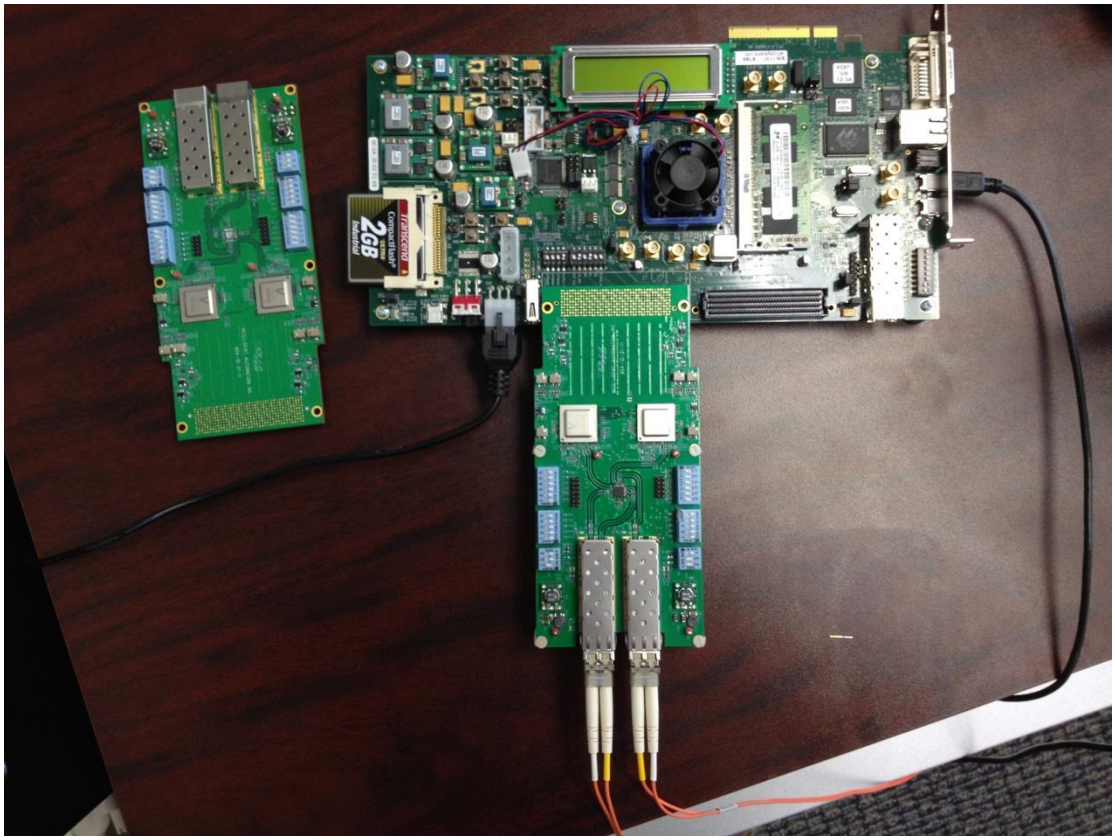

Figure 7 10 Gig Capture PHY and Main board

2 CAGE-100 FPGA Design

Figure 8 shows the analyzer block diagram. The analyzer consists of four major components: PHY Interface; analyzer engine; Memory Management Unit; Host Interface, which are explained below.

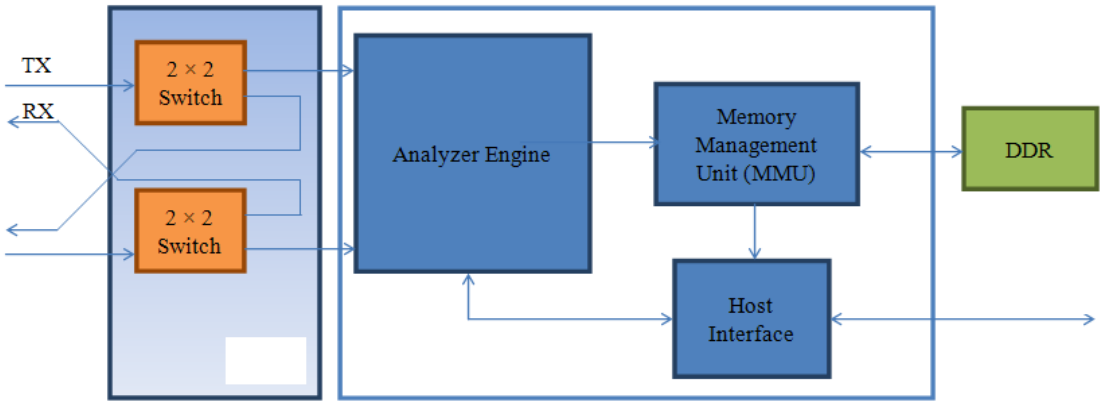

Figure 8. CAGE-100 block diagram.

1. **PHY Interface:** PHY interface is responsible to deliver a copy of data stream to the analyzer engine. For every two nodes under tests, there are two high-speed active fan-out buffers. Each 2 by 2 switch distributes the input TX pair signal into two TX pair signals, one goes to the analyzer and one is routed to RX signal of others node.
2. **Analyzer Engine:** Analyzer engine is designed to capture, filter, and trigger on the incoming data stream and prepare the captured data for MMU unit. The Analyzer engine has a processing unit, Analyzer Processing Unit (APU), which is capable of executing the instructions set by host computer. The instruction sets define what analyzer captures, what is filtered, and when and where the analyzer is triggered.
3. **MMU:** MMU is responsible to store the captured data from analyzer engine and upload the stored data to the analyzer host interface.
4. **Host Interface:** Host Interface communicates with the host computer software via Ethernet to deliver Capture program and other settings to the Analyzer and also upload the capture samples back to the host computer.

2.1 Analyzer Engine

Figure 9 shows a flow chart of CAGE-100 capture tool.

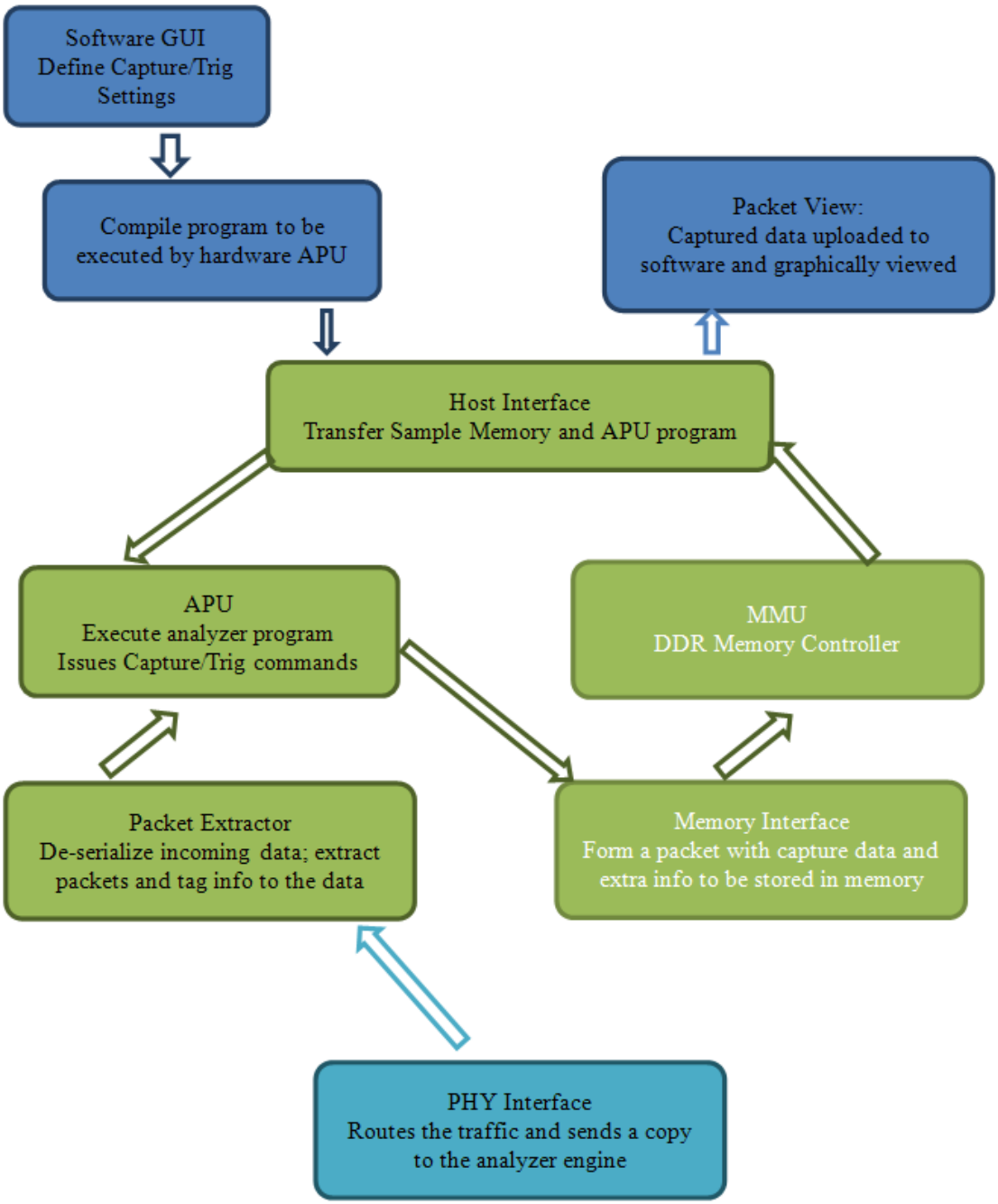

Figure 9 Analyzer Flow Diagram.

Figure 10 shows the block diagram for the analyzer engine.

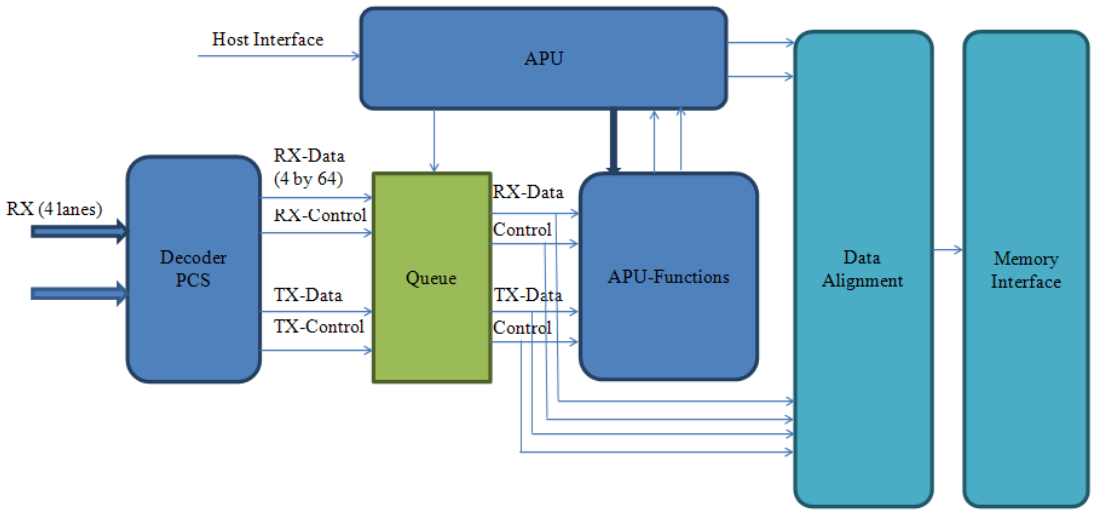

Figure 10 Analyzer block diagram

2.2 Analyzer software

The analyzer software has a Graphical User Interface (GUI) that gets the user capturing strategy and compiles it to a set of instructions. This program is downloaded to the APU memory via Host Interface. All the analyzer settings and options are also programmed in this phase.

2.2.1 How APU works

When APU received the program, it will process the instructions one by one. The program includes the patterns to be captured or to be triggered on for each state of sequencer. There are also a limited number of detector engines, called APU-Functions that are able to detect different patterns in received data. For each analyzer state, APU programs the associated APU-function and wait for the event. When an APU-function event happens, the analyzer state is advanced and APU continues to fetch next instruction and program a new set of APU-functions until the program reaches to the end or the sampling memory becomes full.

2.2.2 How analyzer receives the data from serial data

For both TX lanes of every link, a high speed fan-out buffer is placed on PHY board which is able to repeat a copy of bus data to the analyzer core. This data is fed to serial transceiver to extract the clock, and de-serialize the data. In case of 40GBASE-R, 4 streams of 66-bit data along with associated valid signal are passed to PCS Decoding module.

When the receive channel is in normal mode, the PCS synchronization process continuously monitors the signal lanes. Synchronization module attains block synchronization based on the 2-bit synchronization headers on each one of the PCS lanes. Once block synchronization is found on a PCS lane, then alignment marker lock can be attained by searching for valid alignment markers. After alignment markers are found on all PCS lanes, the PCS lanes can be reordered and de-skewed.

The PCS de-skew process conveys received blocks to the PCS receive process. The PCS deskew process deskews and aligns the individual PCS lanes, removes the alignment markers, forms a single stream, and sets a flag to indicate whether the PCS has obtained alignment.

When the PCS deskew process has obtained alignment, the BER monitor process monitors the signal quality if excessive errors are detected. When align status is asserted the PCS Receive process continuously accepts blocks and generates

Then the data is descrambled with the appropriate code. Once data descrambled the Extractor module detects block type (Control & Data) and add several protocol related information to the data. For example, start and end control characters, payloads and Idles are detected in this module and are aggregated to the sampled data. This process is shown in Figure 11

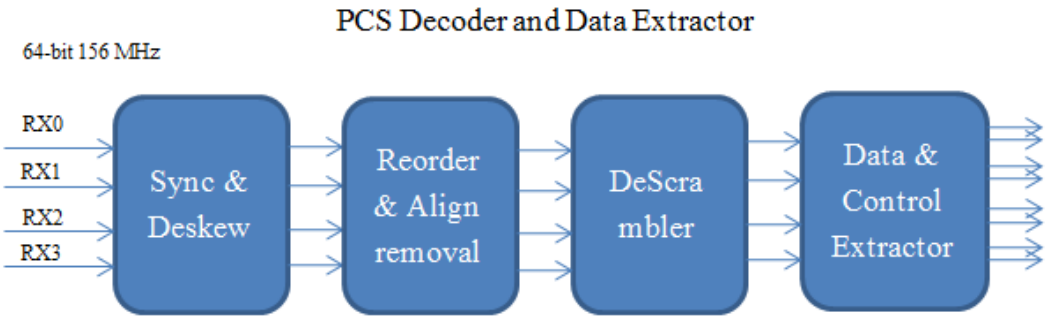

Figure 11 Block diagram of PCS decoder and data & control extractor for 40GBASE-R

The sampled data together with the extracted information is queued up to be fetched by APU. Any time that APU is ready to process incoming data, they fetch a line from queue. To prevent queue overflow, the frequency at which the APU works is slightly higher than the frequency of incoming data.

2.2.3 How captured data is saved into memory

After APU detected the data of interest, it passes them along with the capture command to the memory interface module. This module is responsible to convert raw data samples into sampling memory format. The sampling memory stores captured data samples and some necessary information to be used by software engine. To increase the memory bandwidth efficiency, memory interface removes redundant samples (e.g. Idle octets).Memory Interface sends the data to MMU unit which is connected to the DDR RAM modules. MMU generates DDR commands for saving sampling memory data and passes them to the memory modules.

2.2.4 How data is uploaded into PC

When the sampling memory becomes full or a stop command is received from user, APU reports Host interface to stop data capture. Then, software uploads data from Analyzer. It sends some commands to the Host Interface which is routed to MMU. MMU generates DDR2 commands and read data from memory and passes them to the Host interface. Data is pushed to the Host interface queues and uploaded via Ethernet interface.

2.2.5 APU Main

APU (Analyzer Processing Unit) is responsible for receiving Host Analyzer programming information, programming APU functions unit, Controlling Analyzer sequences & store parameters and finally issuing Store and Trigger commands to the Analyzer engine.

- APU receive analyzer program from software via Host Interface.
- APU starts after receiving start command from Host Interface.
- APU stops after receiving stop command from following sources:
    1. Stop command received from Host Interface.
    2. Stop command received from APU (conditional stop command in advanced analyzer).
    3. Stop command when sampling memory gets full (received from MMU).

The Analyzer state machine supports up to 32 states that in each state:

- Define different capture filter.
- Set Trigger; Stop analyzer.
- Define up to 4 conditions statement. (1 IF and 3 ELSIF, Exit counter)

- Define 32-bit timer.
- Define timeout detector function.
- Two 16-bit Variables that can be ‘Inc’, ‘Dec’ and ‘Load’.

2.2.6 APU Instruction Set

Table below shows the list of instructions that APU executes. The instruction set is programmed by software and sent through Host Interface.

Instruction	Parameters	Description
Pattern	Function, Not, Parameters	Programs a specified function
Start	N/A	Start Analyzer
Stop	N/A	Stop
Condition	Condition#, Count, Mask	Sets special event or events as a term of IF or ElseIf instructions.
Wait	Addr1, Addr2, Addr3, Addr4, Control	Waits on previously defined conditions and jump to Address x if condition x is satisfied.
IncVar	Variable#	Increments variable value once.
DecVar	Variable#	Decrements variable value once
LoadVar	Variable#, Value	Loads an initial value for specified variable.
CJump	Variable#, Value, Address, Type	Compares value of specified Variable with given value and jumps to the given address if comparison result is true. Comparison operator is specified in type field.
Delay	Delay value	

2.2.7 APU Programming

The APU can accept 1024 instructions each of which consists of 16 bytes. An Instruction memory allocated with depth of 1024 and width of 128 bit. Each line contains one machine code except some functions with more than one line parameters (e.g. Data Pattern function which needs at least three lines for its parameters).

Every line is consisted of 16 bytes and every instruction occupies some or all of them. For instructions that do not need all 16 bytes the remained bytes are padded with zero. The machine code for each instruction is listed in the following table:

Instruction	**Machine Code**	
Branch	00	0x02
	01	Address (LSB)
	02	Address (MSB)
Pattern	00	0x03
	01	Function ID (MSB)
	02	Function ID (LSB) odd: N1to N2; even N2 to N1
	03	Bit 0: Not Bit (7…1): Reserved
	04	Bit (7...0): Parameters (7...0)
	05	Bit (7...0): Parameters (15...8)
	:	
	15	Bit (7...0): Parameters (95...88)
Store	00	0x05
	01	Bit0 = 0: Include = 1: Exclude Bit1 = 1: Everything Bit (7...2): Reserved
	02-04	Reserved
	05	Bit (7...0): Store Mask of F0 to F7
	...	
	15	Bit (7...0): Store Mask of F80 to F87
Trig	00	0x06
Exclude	00	0x07
	01	Exclude Mask (7...0)
	…	
	06	Exclude Mask (63...56)
Loop	00	0x08
	01	Address (LSB)

	02	Address (MSB)
	03	Counter (LSB)
	04	Counter (MSB)
Stop Analyzer	00	0x0A
Condition	00	0x0B
	01	Bit (1...0) = Condition# Bit (3...2) = Reserved Bit (7...4) = Count (LSB)
	02-03	Bit (11...0) = Count (MSB) Bit (15...12) = Reserved
	04	Bit (2...0): Condition Mask of T1 to T3 (used for Timer Enable) Bit (3): Condition Mask of Anything Bit (4): Condition Mask of Timeout Bit (7...5): Condition Mask are Reserved
	05	Bit (7...0): Condition Mask of F0 to F7
	…	
	15	Bit (7...0): Condition Mask of F80 to F87
Wait	00	0x04
	01	Addr1 (LSB)
	02	Addr1 (MSB)
	03	Addr2 (LSB)
	04	Addr2 (MSB)
	05	Addr3 (LSB)
	06	Addr3 (MSB)
	07	Addr4 (LSB)
	08	Addr4 (MSB)
	09	Control (7...0)
	10	Control (15...8)
	11	Control (23...16)
	12	Control (31...24)

IncVar	00	0x0C
	01	Bit 0 = Variable#
DecVar	00	0x0D
	01	Bit 0 = Variable#
LoadVar	00	0x0E
	01	Bit 0 = Variable# Bit (7...1) = Reserved
	02	Value (LSB)
	03	Value (MSB)
CJump	00	0x0F
	01	Bit 0 = Variable# Bit (7…1) = Reserved
	02	Value (LSB)
	03	Value (MSB)
	04	Address (LSB)
	05	Address (MSB)
	06	Bit (1...0) = Type Bit (7...2) = Reserved
Start Analyzer	00	0x10
Delay	00	0x11
	01	Delay (LSB)
	02	Bit (3…0) = Delay (MSB) Bit (7…4) = Reserved
Halt	00	0xFF

2.2.8 Instructions Description

2.2.8.1 Pattern {Function#, Not, Parameters}

With Pattern instruction, one can define various types of pattern within Ethernet frame to be detected. There's a 1-bit field that defines whether the instruction should find or ignore the pattern.

There are three global 32-bit timers and one timeout timer are available which can set in each state. In order to program these timers Pattern instruction must be used with function ID of 0x01, 0x02, 0x03, and 0x04. The first three IDs are for the three timers and the fourth one is for timeout timer. Also "Parameters" field specifies Timer Value T.

Using Pattern instruction, we can define various types of patterns, symbols, protocol errors and etc. In some cases, Parameters field is too big be defined in one instruction, therefore multiple Pattern instruction maybe used to define long patterns. For long patterns, i.e. more than 16bytes, the last parameter byte defines the instruction number with that pattern and the last Pattern instruction number is FF.

2.2.8.2 Start Analyzer

This instruction starts the analyzer.

2.2.8.3 Stop Analyzer

This instruction stops the analyzer.

2.2.8.4 Trig {No Parameters}

This instruction is used to trigger the analyzer.

2.2.8.5 Exclude

This instruction excludes the specified pattern from being captured. Format of APU Exclude command mask for 10GBASE-R is as follows:

Exclude Mask	**Position**
Exclude Data Frame Payload	Bit 0
Exclude Idle	Bit 1
Exclude START /S/	Bit 2
Exclude TERMINATE /T/	Bit 3
Exclude ERROR /E/	Bit4
Exclude SEQUENC ORDERED_SET /Q/	Bit5
Reserved	Bit6
Reserved	Bit7
Reserved	Bit8
Exclude Payload Offset [11..0]	Bit 27 to 16

Exclude Payload Offset indicates offset from begin of payload that should be excluded.

2.2.8.6 Jump {Address}

Unconditionally branches to the next instruction address value.

2.2.8.7 Loop {Label, Counter}

This instruction jumps to the Label if the Counter is not zero, then decrements the Counter by one. Label field defines the next branch address.

2.2.8.8 CJump {Variable#, Value, Address, Type}

This instruction compares the value of specified variable with specified *Value* and branches to any other state if the comparison is true. The *Address* field of *CJump* instruction sets branch address and *Type* field sets comparison operator type.

Type field format is as follows:

00: =

01: > (Variable >Value)

10: < (Variable <Value)

11: Reserved

There are two variables that could be selected by setting or resetting of "Variable" field of "CJump" Instruction.

2.2.8.9 LoadVar {Variable#, Value}

This instruction loads initial *Value* for specified variable. The *Variable#* field of *LoadVar* instruction selects the variables.

2.2.8.10 IncVar {Variable#}

This instruction increments the variable.

2.2.8.11 DecVar {Variable#}

This instruction decrements variable value.

2.2.8.12 Condition {Condition #, Count, Mask}

Analyzer supports four IF and ELSIFs within each state. This instruction defines condition of IF and ELSIFs. *Condition#* field indicates the ID of used conditions (Condition1 (IF) with *Condition#* = 00, Condition2 (ELSIF1) with *Condition#* = 01, Condition3 (ELSIF2) with *Condition#* = 10 or Condition4 (ELSIF3) with *Condition#* = 11). Using *Condition#* field, we can refer to different *Condition* instructions in each state (e.g. in *Control* field of *Wait* instruction).

Also each condition parameters that are searched by analyzer is marked with *Mask* bits. Note that Condition Mask bits of 0 to 2 are used for timer enabling, bit 3 is used for *Anything* and bit 4 is used for *Timeout*. Higher bytes of instruction are used for masking F0 to F87.

In each condition one 16-bit *event counter* can be set. The *Count* field is allocated for this counter. In this case, *Mask* bits show event or events, which their occurrence must be counted.

For each IF and ELSIF used in each state one signal *Condition* Instruction must be used.

2.2.8.13 Wait {JumpAddr1, JumpAddr2, JumpAddr3, JumpAddr4, Control}

This instruction waits on previously defined conditions and jumps to "*JumpAddressX*" if condition number X is true. Additional "Control" field puts control over "Wait" conditions. "Control" field defines next "*JumpAddress*" or any other transition. It consists of:

Bit0: Condition1 is active (and so "JumpAddress1" is valid)
Bit1: Condition2 is active (and so "JumpAddress2" is valid)
Bit2: Condition3 is active (and so "JumpAddress3" is valid)
Bit3: Condition4 is active (and so "JumpAddress4" is valid)

Bit4: Set Trigger after condition 1 is met
Bit5: Set Trigger after condition 2 is met
Bit6: Set Trigger after condition 3 is met
Bit7: Set Trigger after condition 4 is met

Bit8: Stop Analyzer after condition 1 is met
Bit9: Stop Analyzer after condition 2 is met
Bit10: Stop Analyzer after condition 3 is met
Bit11: Stop Analyzer after condition 4 is met

Bit12: Set External Signal after condition 1 is met
Bit13: Reset External Signal after condition 1 is met
Bit14: Set External Signal after condition 2 is met
Bit15: Reset External Signal after condition 2 is met
Bit16: Set External Signal after condition 3 is met
Bit17: Reset External Signal after condition 3 is met
Bit18: Set External Signal after condition 4 is met
Bit19: Reset External Signal after condition 4 is met

Bit20: This bit should be set if condition1 is used but it does not jump to other states
Bit21: This bit should be set if condition2 is used but it does not jump to other states
Bit22: This bit should be set if condition3 is used but it does not jump to other states

Bit23: This bit should be set if condition4 is used but it does not jump to other states

2.2.8.14 Outport {Address, Value}

This instruction is used to set some output register including external signal and state indication. Address field specifies registers address and Value indicates the value of that register.

Address 0x00h is used for External signals. In this case Bit 0 of Value shows external signal, other bits are reserved. Address 0x01h is used for state indication. Software could mark state IDs by using this instruction with address 0x01h and proper value, and use it during run process in order to indicate which state is being executed (e.g. (outport 1,2) means in this time state 2 is executed). The addresses 0x02 to 0x0F are reserved for APU registers.

2.2.8.15 Store {Control, Mask}

This Instruction is used to capture specified pattern. Control field defines that selected patterns should be captured or filtered out.

2.2.8.16 Halt {No Parameters}

This instruction indicates the end of program.

2.2.8.17 Delay {Delay}

This instruction generates three or more clocks delay in an APU program and mainly can be used to synchronize multiple APU cores in cascade architecture. The resulted delay by the Delay instruction is “delay value” + 3 clocks. This means the delay parameter should be set to three less than the desired value.

2.2.9 APU programming examples

2.2.9.1 Example 1

In this example, we capture everything and trig at the beginning

Capture

- Everything

Trig-On

- Snapshot

Memory

- Entire Memory
- Pre-Trig = 50%

No	Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	start	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2	Outport	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3	Outport	01	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00
4	trig	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
5	exclude	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
6	store	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7	halt	FF	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

2.2.9.2 Example 2

Capture

- Pattern
    1. ARP Frame on both direction; N1 to N2 , N2 to N1
        - Source MAC: 14:FE:B5:D9:1F:FF,
        - ARP Type: “0806”,
        - Target IP Address: 10:5:1:252

Trig-On

- Snapshot

Trace Memory

- Entire Memory
- Pre-Trig = 50%

No	Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	Pattern	03	00	01	00	00	00	00	00	00	00	14	FE	B5	D9	1F	00
2	Pattern	03	00	01	00	FF	88	66	00	00	00	00	00	00	00	00	01
3	Pattern	03	00	01	00	00	00	00	00	00	00	00	00	00	00	00	02
4	Pattern	03	00	01	00	00	A5	51	FC	00	00	00	00	00	00	00	03
5	Pattern	03	00	01	00	00	00	00	00	00	00	00	00	00	00	00	04
6	Pattern	03	00	01	00	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
7	Pattern	03	00	02	00	00	00	00	00	00	00	14	FE	B5	D9	1F	00
8	Pattern	03	00	02	00	FF	88	66	00	00	00	00	00	00	00	00	01
9	Pattern	03	00	02	00	00	00	00	00	00	00	00	00	00	00	00	02
10	Pattern	03	00	02	00	00	A5	51	FC	00	00	00	00	00	00	00	03
11	Pattern	03	00	02	00	00	00	00	00	00	00	00	00	00	00	00	04
12	Pattern	03	00	02	00	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
13	start	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
14	Outport	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
15	Outport	01	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00
16	trig	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
17	exclude	07	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1	store	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

8		2	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
1	halt	F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9		F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2.2.10 APU Functions

This component is responsible to detect patterns defined by APU through following functions:

- Control Character detection
- Frame Pattern detection
- Data pattern detection
- Protocol error detection

As shown in Figure 12 Function Decoder decodes and enables appropriate function based on what it is received from APU. Then any selected function (except Timer & External Trigger functions) starts to compare data stream (received from *Data Queue*) with function parameters (received from *APU*) and if they were matched, respected function would set *Store* and *Trig* Signals to '1' for one clock pulse period. To accomplish this process each function utilizes some control signals (frame Start/Terminate, block type ...) from *Data Queue* and from another component within *APU Function* called *Frame Extraction* which gives additional information about current frame (MAC address, frame type …).

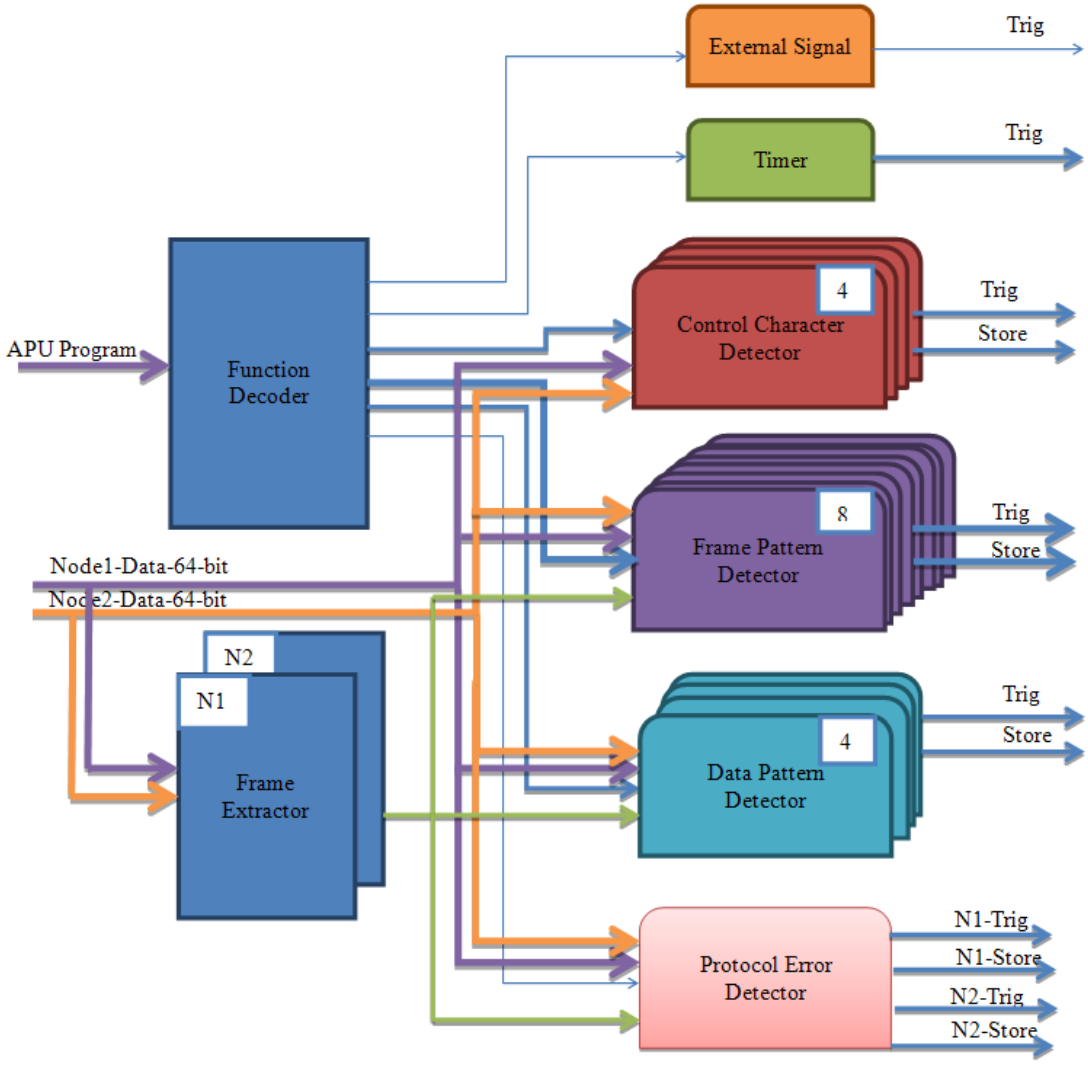

Figure 12 APU Function block diagram

2.2.11 Data Queue

This component is comprised of an asynchronous FIFO and a controller that can temporarily store SerDes extracted data stream and their relative control signals to be further used by Analyzer engine (APU, APU functions). This queue compensates for the inherent delay existed in fetch & decoding process of APU instructions. Figure 13 shows Data Queue block diagram.

By receiving start analyzer command from APU, store information are written into FIFO with the 156.25 MHz clock signal and when read enable issued by APU, APU Functions reads stored information with 200 MHz clock signal.

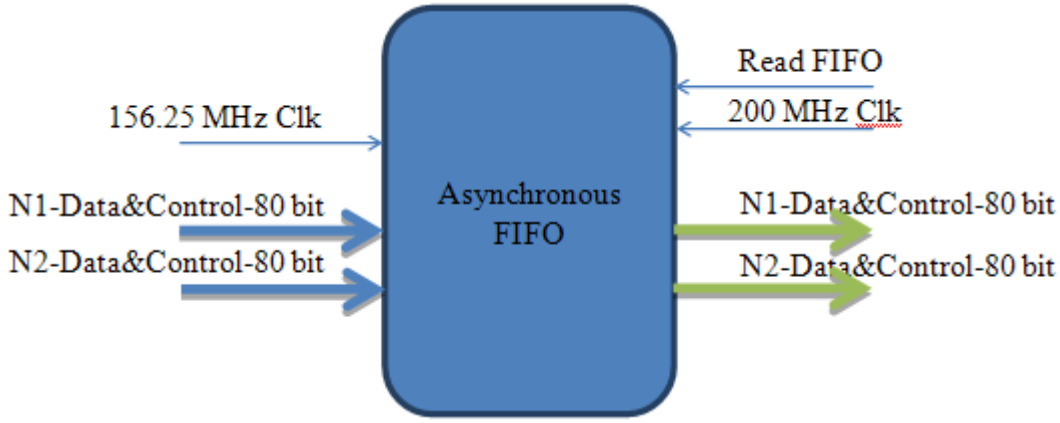

Figure 13 Data Queue block diagram

2.2.12 Data Alignment

"Data Alignment" component consists of series of signal pipes that adjusts output signals of "Data Queue" component to the issued commands of APU (trig, store commands …) and protocol errors outputs of "APU Functions". This alignment is necessary when these two sets of information are further used in construction of sampling memory where any captured word or frame must be co-sited with its pertaining protocol error and/or trig/store information.

2.2.13 Memory Interface

Memory Interface constructs the analyzer sampling memory format in which stores SerDes derived data and some additional information like time stamp, duration and protocol errors. *Memory Interface* resets with a short delay (80 ns) after receiving start command from APU. There are two instances of *Memory Interface*s, each of which stores single node information. Depending on type of data we have three formats to be stored in sampling memory.

1. Idle sequence that consists of idle control characters which exist outside frame sequence.
2. Control character sequence that consists of control characters.
3. Frame sequence that consists of all Ethernet supported frames. (e.g. ARP, UDP,TCP)

These sequences are placed in a row of 128-bit data to store two DWORDs plus some additional information (C/D, symbol error...). Then at the end of each sequence, an extra line with information about time stamp, duration and protocol errors is formed. Each sequence belongs to one direction data stream. While N1 & N2 sides of *Memory*

Interface is constructing sampling memory data there is a state machine that sort the received memory data and extra line according to their start time in which any sequence from either side that starts prior to the opposite side will be sent to MMU (Memory Management Unit) earlier. Along with Memory data and extra line, an eight-bit command containing information about (start and end sequence, trig command and extra line valid command) sends to the MMU.

In each *Memory Interface* instance, after constructing memory format data are temporarily stored in three separate FIFOs (data, complement line, time stamp) in order to be further read by sorting state machine. These FIFOs compensate for intrinsic delay that exists through state machine transitions.

Figure 14 shows Memory Interface block diagram.

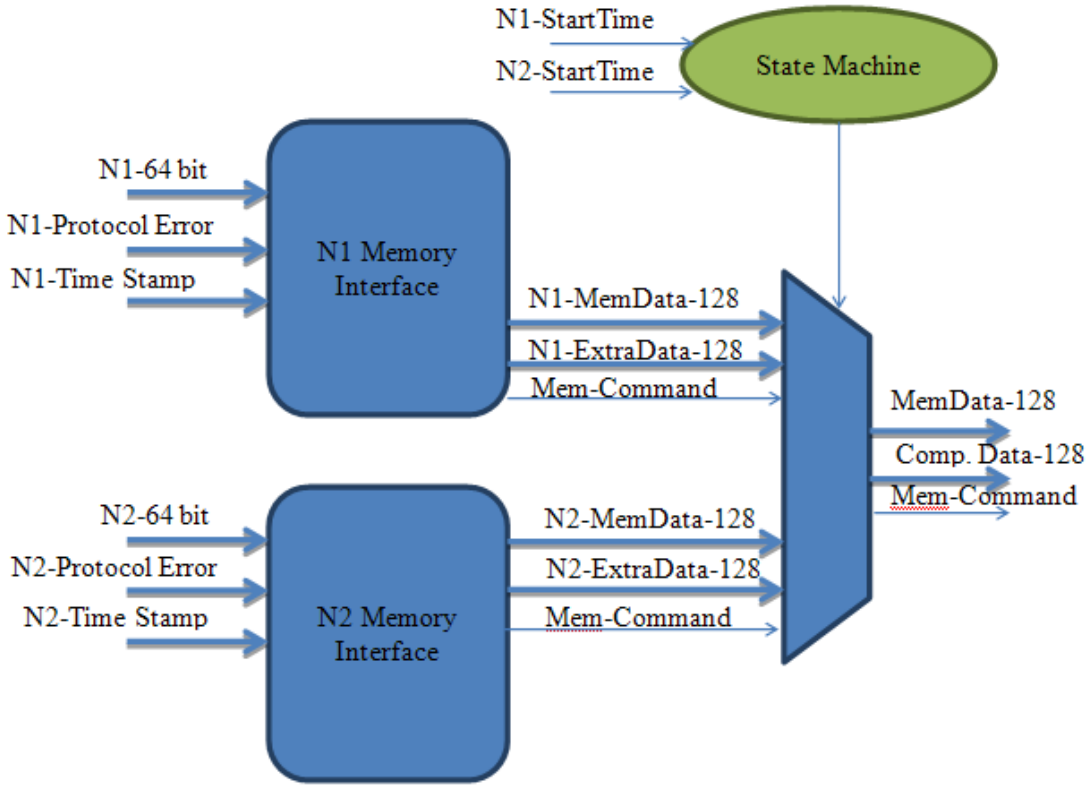

Figure 14 Memory Interface block diagram

2.2.14 Frame Sequence (First and Middle rows)

Frame sequence is a sequential of data that starts with Start control character /S/ and ends with Terminate control character /T/. Each frame sequence occupies some rows of memory. Each row stores a block of 64-bit data plus the block extra information. Format of rows except for the last row is as defined in Figure 15:

Byte 15-14	Byte13-10	Byte 9-2	C/D	Byte 0
Control Char Type	Flags	Block	Control/Data	Symbol Err

Figure 15- Format of first row and middle rows of Frame sequence in sampling memory

Each frame sequence consists of a first row; some middle rows with overall format similar to first row and a last row. To detect a frame sequence, at first these rows should be recognized. Bit 11.1:0 should be checked to identify the row and then bit 11-3:2 to find out what type of sequence i.e. frame or idle, this row belongs to.

Symbol Err (Byte 0): This field is allocated for symbol error indication. If the received octet has symbol error, i.e. descramble error or invalid control character, the corresponding bit in Byte 0 is set to '1'.

Each bit of this field indicates status of one symbol. 1 means the associated symbol has error.

Byte 0	Description
Bit-0	Octet0 Error
Bit-1	Octet1 Error
Bit-2	Octet2 Error
Bit-3	Octet3 Error
Bit-4	Octet4 Error
Bit-5	Octet5 Error
Bit-6	Octet6 Error
Bit-7	Octet7 Error

(Byte 1): Type of data octet is defined in this field. '0' for Data octet and '1' for control character octets.

(Byte 9-2): One block of data is stored in this field. Byte 10 stores D0 (Least Significant Byte) and Byte 3 stores D7 (Most Significant Byte).

(Byte 13-10): These four bytes are reserved for flags.

Byte 10-13	Description
Bit-10.0	Octet0 is Align that removed/inserted for clock correction
Bit-10.0	Octet1 is Align that removed/inserted
Bit-10.1	Octet2 is Align that removed/inserted
Bit-10.2	Octet3 is Align that removed/inserted
Bit-10.3	Octet4 is Align that removed/inserted
Bit-10.4	Octet5 is Align that removed/inserted
Bit-10.5	Octet6 is Align that removed/inserted
Bit-10.6	Octet7 is Align that removed/inserted
Bit-10.7	RSV
Bit-11.1:0	"00" : First row
	"01" : middle row
	"10" : Last row
	"11" : RSV
Bit-11.3:2	"00" Frame Seq
	"01" Idle Seq
	"10" RSV
	"11" RSV
Bit-11.4	Start of frame:Has /S/
Bit-11.5	End of frame: Has /T/
Bit-11.6	Protocol Error detected on Octet0
Bit-11.7	Protocol Error detected on Octet1
Bit-12.0	Protocol Error detected on Octet2
Bit-12.1	Protocol Error detected on Octet3
Bit-12.2	Protocol Error detected on Octet4
Bit-12.3	Protocol Error detected on Octet5
Bit-12.4	Protocol Error detected on Octet6
Bit-12.5	Protocol Error detected on Octet7
Bit-12.6	'0' means octet0 excluded
Bit-12.7	'0' means octet1 excluded
Bit-13.0	'0' means octet2 excluded
Bit-13.1	'0' means octet3 excluded

Bit-13.2	'0' means octet4 excluded
Bit-13.3	'0' means octet5 excluded
Bit-13.4	'0' means octet6 excluded
Bit-13.5	'0' means octet7 excluded
Bit-13.6	RSV
Bit-13.7	RSV

(Byte 15-14): Reserved.

2.2.15 Frame sequence (Last row)

The complement data of memory format for frame sequences places at the last row and the format is shown in Figure 16:

Byte 15-14	Byte 13-12	Byte 11	Byte 10-9	Byte 8-4	Byte 3-0
External Signal In	Flags	Extended Flags	Protocol Error	Start Time (5 bytes)	Duration (4 bytes)

Figure 16 Last Row Format

Duration (Byte 3-0): Duration of frame sequence, this value should be divided by clock frequency to get duration in second. The clock frequency is 156.25e6 for 10GBASE-R.

Start Time (Byte 8-4): This value should be divided by clock frequency, i.e. 156.25e6 to calculate start time in second.

Protocol Error (Byte 10-9): List of specified protocol errors:

Byte 10-9	Description
Bit-9.0	RSV.
Bit-9.1	RSV.
Bit-9.2	RSV.
Bit-9.3	RSV.
Bit-9.4	RSV.
Bit-9.5	RSV.
Bit-9.6	RSV.
Bit-9.7	RSV.
Bit-10.0	RSV.

Bit-10.1	RSV.
Bit-10.2	RSV.
Bit-10.3	RSV.
Bit-10.4	RSV.
Bit-10.5	RSV.
Bit-10.6	RSV.
Bit-10.7	RSV.

Flags (Byte 11):

Byte 11	Description
Bit-11.1:0	"00" : First row
	"01" : middle row
	"10" : Last row
	"11" : RSV
Bit-11.3:2	"00" Frame Sequence
	"01" Idle Sequence
	"10" RSV
	"11" RSV
Bot-11.7:4	APU state

Flags (Byte 13-12):

Byte 13-12	Description
Bit-12.0	RSV.
Bit-12.1	RSV.
Bit-12.2	Frame direction: '0' N1 to N2; '1' N2 to N1
Bit-12.3	Payload status: '0' stored ; '1' Not stored
Bit-12.4	Scrambling status: '0' de-scrambled; '1' not de-scrambled
Bit-12.5	Frame protocol type:
Bit-12.6	"00000" : ARP
Bit-12.7	"00001" : UDP
Bit-13.0	"00010": TCP

Bit-13.1	
Bit-13.2	'1' frame not completely captured, Memory full
Bit-13.3	Split frame: "00" : not split "01" : first split frame "10" : middle frame "11" : last frame
Bit-13.4	
Bit-13.5	RSV.
Bit-13.6	RSV.
Bit-13.7	RSV.

External Signal In: External signals value. External Signals could be sampled at the beginning of a sequence. In the last row of each sequence, this value is stored.

2.2.16 Idle Sequence Format

The data octets that come outside Frame sequence are stored in an *Idle* sequence format. Instead of storing the exact value of idle octets, the number of valid idle octets is stored. If the idle octet is an invalid code, the exact code is stored in the format of frame sequence. Figure 17 shows the Idle sequence format.

Byte 15-14	Byte13-10	Byte 9-2	Byte 1	Byte 0
Control Char Type	Flags	Number of idle octets	RSV	Symbol Err

Figure 17 Idle Sequence Format

Symbol Err (Byte 0): This field is allocated for symbol error indication. If the received idle octet has symbol error, i.e. descramble error or invalid control character, the corresponding bit in Byte 0 is set to '1'.
Each bit of this field indicates status of one symbol. 1 means the associated symbol has error.

Byte 0	Description
Bit-0	Octet0 Error
Bit-1	Octet1 Error
Bit-2	Octet2 Error
Bit-3	Octet3 Error
Bit-4	Octet4 Error
Bit-5	Octet5 Error
Bit-6	Octet6 Error
Bit-7	Octet7 Error

(Byte 1): Reserved.

(Byte 9-2): The number of idle data octets is stored in Byte 3:2, i.e. LSB byte 2 MSB Byte 3. If there's any invalid idle octet the exact value is stored, i.e. Byte 10 stores D0 (Least Significant Byte) and Byte 3 stores D7 (Most Significant Byte).

(Byte 13-10): These four bytes are reserved for flags.

Byte 10-13	Description
Bit-10.0	Octet0 is Align that removed/inserted for clock correction
Bit-10.0	Octet1 is Align that removed/inserted
Bit-10.1	Octet2 is Align that removed/inserted
Bit-10.2	Octet3 is Align that removed/inserted
Bit-10.3	Octet4 is Align that removed/inserted
Bit-10.4	Octet5 is Align that removed/inserted
Bit-10.5	Octet6 is Align that removed/inserted
Bit-10.6	Octet7 is Align that removed/inserted
Bit-10.7	RSV
Bit-11.1:0	"00" : First row
	"01" : middle row
	"10" : Last row
	"11" : RSV
Bit-11.3:2	"00" Frame Sequence
	"01" Idle Sequence
	"10" RSV
	"11" RSV

Bit-11.4	RSV
Bit-11.5	RSV
Bit-11.6	Protocol Error detected on Octet0
Bit-11.7	Protocol Error detected on Octet1
Bit-12.0	Protocol Error detected on Octet2
Bit-12.1	Protocol Error detected on Octet3
Bit-12.2	Protocol Error detected on Octet4
Bit-12.3	Protocol Error detected on Octet5
Bit-12.4	Protocol Error detected on Octet6
Bit-12.5	Protocol Error detected on Octet7
Bit-12.6	'0' means octet0 excluded
Bit-12.7	'0' means octet1 excluded
Bit-13.0	'0' means octet2 excluded
Bit-13.1	'0' means octet3 excluded
Bit-13.2	'0' means octet4 excluded
Bit-13.3	'0' means octet5 excluded
Bit-13.4	'0' means octet6 excluded
Bit-13.5	'0' means octet7 excluded
Bit-13.6	RSV
Bit-13.7	RSV

(Byte 15-14): Reserved.

2.2.17 Idle sequence (Last row)

The last row of memory format for frame sequences is shown in Figure 18:

Byte 15-14	Byte 13-12	Byte 11	Byte 10-9	Byte 8-4	Byte 3-0
External Signal In	Flags	Extended Flags	Protocol Error	Start Time (5 bytes)	Duration (4 bytes)

Figure 18 Last Row Format

Duration (Byte 3-0): Duration of frame sequence, this value should be divided by clock frequency to get duration in second. The clock frequency is 156.25e6 for 10GBASE-R.

Start Time (Byte 8-4): This value should be divided by clock frequency, i.e. 156.25e6 to calculate start time in second.

Protocol Error (Byte 10-9): List of specified protocol errors.

Byte 10-9	Description
Bit-9.0	RSV.
Bit-9.1	RSV.
Bit-9.2	RSV.
Bit-9.3	RSV.
Bit-9.4	RSV.
Bit-9.5	RSV.
Bit-9.6	RSV.
Bit-9.7	RSV.
Bit-10.0	RSV.
Bit-10.1	RSV.
Bit-10.2	RSV.
Bit-10.3	RSV.
Bit-10.4	RSV.
Bit-10.5	RSV.
Bit-10.6	RSV.
Bit-10.7	RSV.

Flags (Byte 11):

Byte 11	Description
Bit-11.1:0	"00" : First row
	"01" : middle row
	"10" : Last row
	"11" : RSV
Bit-11.3:2	"00" Frame Sequence
	"01" Idle Sequence
	"10" RSV
	"11" RSV
Bot-11.7:4	APU state

Flags (Byte 13-12):

Byte 13-12	Description
Bit-12.0	RSV.
Bit-12.1	RSV.
Bit-12.2	Frame direction: '0' N1 to N2; '1' N2 to N1
Bit-12.3	RSV
Bit-12.4	RSV
Bit-12.5	RSV
Bit-12.6	
Bit-12.7	
Bit-13.0	
Bit-13.1	
Bit-13.2	RSV
Bit-13.3	Split frame: "00" : not split "01" : first split frame "10" : middle frame "11" : last frame
Bit-13.4	
Bit-13.5	RSV.
Bit-13.6	RSV.
Bit-13.7	RSV.

External Signal In: External signals value. External Signals could be sampled at the beginning of a sequence. In the last row of each sequence, this value is stored.

2.3 Memory Management Unit

2.3.1 Overview

There are two modules that have access to the memory buffer. The *Analyzer Engine* that captures data and *Host Interface* that receives capture data to showing it in the viewer. *Memory Management Unit* (MMU) controls these data transactions from analyzer engine to DDR Memory and from *Host Interface* to DDR Memory. Figure 19 shows the block diagram for MMU with respect to the whole system.

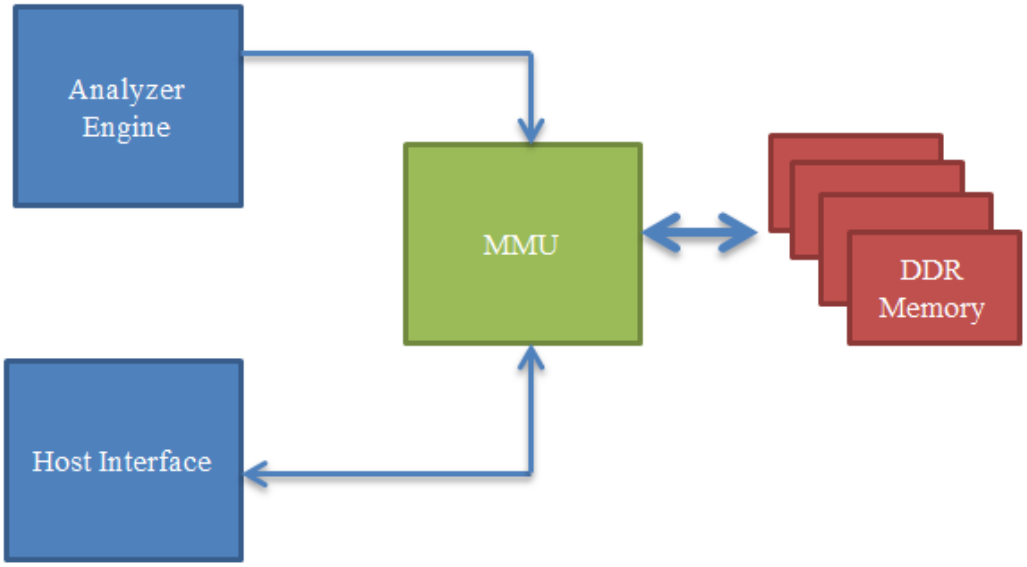

Figure 19 Memory Management Unit

Inside MMU there are three components:

2.3.2 Analyzer Command Extractor (ACE)

The *Analyzer Engine* doesn't provide the write address; rather it only generates some commands. The MMU extracts the commands in order to generate the address and write requests. This module is called *A*nalyzer Command Extractor (ACE).

2.3.3 Host Interface Management (HIM)

Host Interface Management controls the data transactions between memory and *Host Interface.*

2.3.4 DDR Memory Multiplexer (DMM)

DMM receives incoming requests from ACE and HIM and constructs the write/read request to the DDR3 and deliver the read data to HIM. Figure 20 shows the block diagram for MMU.

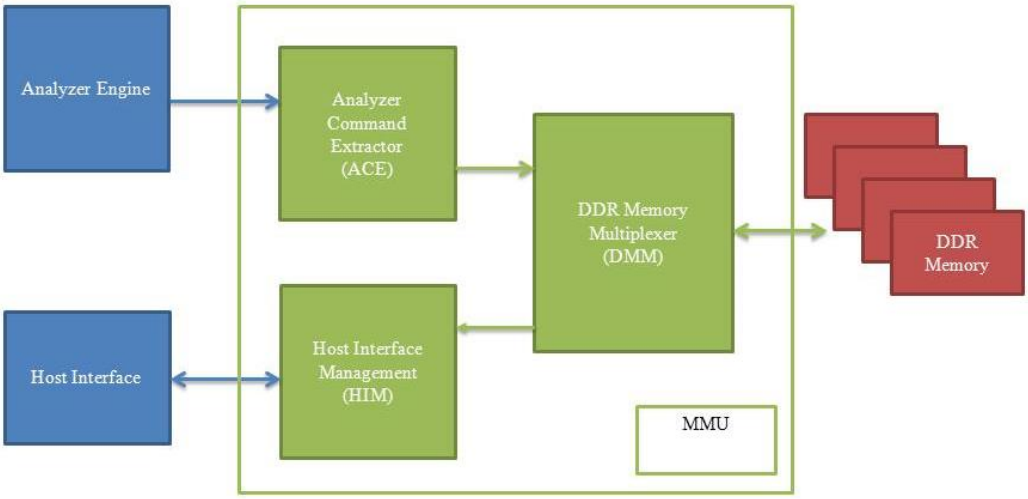

Figure 20: MMU block diagram

In the next subsections, you can find a brief explanation about duties of each part.

2.3.4.1 ACE

The Analyzer Engine provides ACE with data prepared by *Memory Interface*, i.e. sampled data, extra information, protocol error and time stamp. Also store and trig commands are passed from Analyzer Engine to ACE. ACE receives the data and commands and prepares the DDR commands and data.

Figure 21 shows the block diagram for the ACE module.

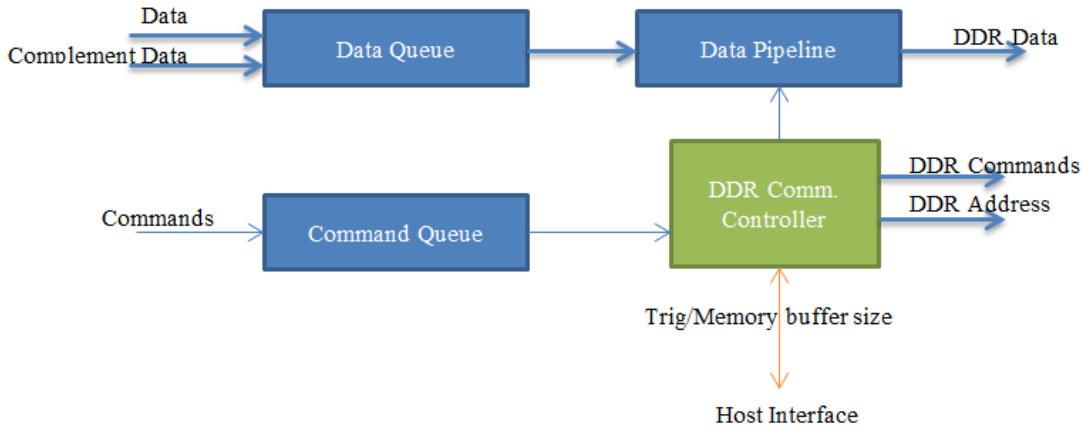

Figure 21: Analyzer Command Extractor

In order to keep track of stored sample size, ACE communicates with Host Interface. Host Interface can inform ACE with memory sample and Pre/Post trig size. ACE also in turn can report to Host Interface with memory status, i.e. stop and trig addresses.

Incoming command vector that comes from Analyzer Engine (Memory Interface) are as follows:

Index	Name	Description
0	Analyzer Start	Asynchronous reset for address counter and address Registers (Tag, End and Trig address registers).
1	Analyzer Stop	Resets ACE logics.
2	Start of Packet	Indicates start of a packet.
3	End of Packet	Indicates the end of a packet.
4	Store Packet	Indicates the current packet must be stored
5	Trigger	Signal the post trigger counter to start counting.
6	Store Mode	'0' Store packets; '1' Exclude packets
7	Discard Line	Indicates the current line is discarded

Table 1 ACE input commands

2.3.5 HIM

All accesses of the host computer to the storage memory will be done through the host interface unit. Host interface execute HAL commands.

Host Interface communicates with host using a 16 bits bi-directional data bus. Three devices use this data bus: Host, MMU (using host interface) and APU. Each device has two status bits that show the device status. Four states have been defined for each device: IDLE, SEND, RECEIVE and WAIT. The state of a device is known using its status bits as follows:

Status	Status bits
IDLE	00
SEND	01
RECEIVE	10
WAIT	11

Table 2 HIM Bus Status

Each device can use data bus for sending its data only when the other two are in IDLE state. Four instructions have been defined for host interface unit till now:

1. Memory read
2. Get memory status (essential address read)
3. Set memory setting (sample number and post sample number)
4. Get memory configuration.

Each instruction has a one-word (16-bits) code and up to four word parameter. Instruction codes are as follows:

Instruction	Code
Memory read	0701
Set memory setting	0702
Get memory status	0703
Get memory configuration	0704

Table 3 MMU Commands

2.3.5.1 Memory read

Host interface reads the requested area from storage memory, and sends them to the host. Instruction format is as follows:

15	0
Instruction Code (0701)	
Start Address (15…0)	
Start Address (23 … 16)	
End Address (15 … 0)	
End Address (23 … 16)	

The first word is instruction code. The second word is the 16 least significant bits of the start address. The third word contains the 8 most significant bits of the start address. The fourth word is the 16 least significant bits of the end address. The fifth word contains the 8 most significant bits of the end address. After receiving this instruction, the host interface will return the memory area from start address till end address.

After receiving instruction, host interface will remain in receive state until data read from memory being ready to send.

The size of the packet is adjustable. Host interface will go to IDLE state after sending the last packet otherwise (after sending the middle packets or first packet) it will go to

WAIT state. Each packet sending from host interface to host has a 1-word header. Following Table shows the header of the packets.

Packet	Header
First packet	001C
Middle packet	011C
Last packet	021C

Table 4: Header of the packets send from HIC to Host

2.3.5.2 Get memory status

Host interface read the essential registers contents and send them to the host. This instruction does not have any parameter. Instruction format is as follows:

15	0
Instruction Code (021D)	

After receiving this instruction, the host interface will return the content of Start-address register, Current-address register, End-address register, Trig-Address register, Trig-time register and Status register.
The packet returned from host interface has been shown below:

15	0
Header	
RX Bank Start-Add (15 … 0)	
RX Bank Start-Add (23 … 16)	
RX Current-Add (15 … 0)	
RX Current-Add (23 … 16)	
RX End-Add (15…0)	
RX End-Add (23…16)	
RX Status	
TX Bank Start-Add (15 … 0)	

TX Bank Start-Add (23 … 16)
TX Current-Add (15 … 0)
TX Current-Add (23 … 16)
TX End-Add (15…0)
TX End-Add (23…16)
TX Status
Trig time stamp (15…0)
Trig time stamp (31…16)
Trig time stamp (39…32)

Header of this packet is “021E”. Timing diagram of the instruction is shown in

2.3.5.3 Set memory setting

Host interface receive the sample number and post sample number values from host and write them in sample-no and post-sample-no registers. The instruction format is as follows:

15	0
Instruction Code (0702)	
Sample No (15…0)	
Sample No (31 … 16)	
Post Sample No (15…0)	
Post Sample No (31 … 16)	

2.3.5.4 Get memory configuration

This instruction returns constant value. The instruction format and the packet returned by host interface are shown below.

15	0
Instruction Code (0704)	

15	0
Header (0212)	
0001	

2.3.5.5 DMM

DDR Memory Multiplexer (DMM) is a data and command multiplexer for DDR memory. ACE and HIM memory requests go through DMM.

There are two independent sources that can generate access requests, Analyzer and Host Interface. Analyzer only asks for write to memory. Host Interface asks for both read and write. DMM is responsible of collecting different requests, prioritizing them and forwarding them all to the DDR2 controller.

2.3.5.6 DDR3 Controller

For DDR3 Controller Xilinx MIG memory controller is used.