UCL

Upstream DAQ Technology Review
10s Buffer Management Firmware

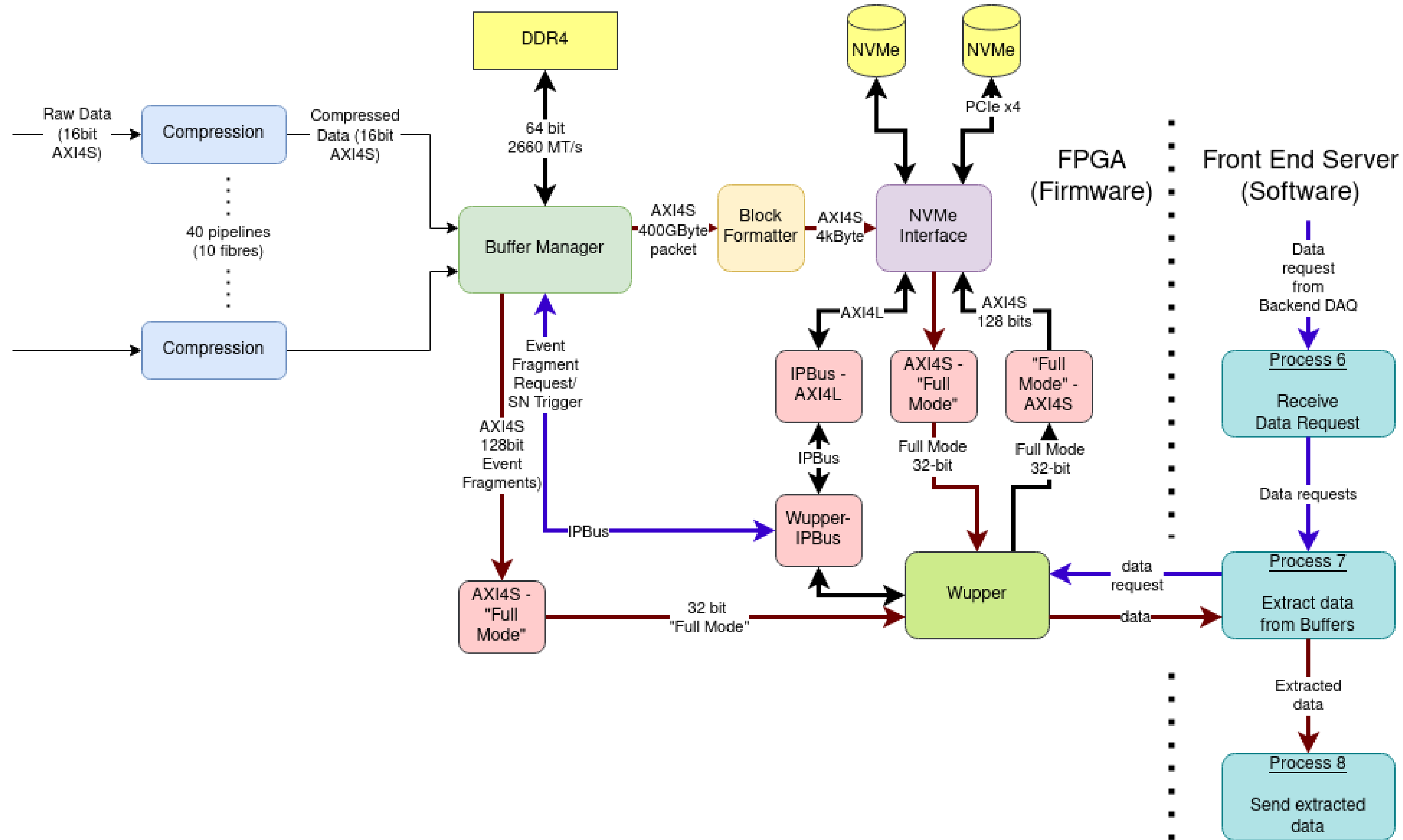
Erdem Motuk

29/07/2020

Outline

- System-level view
- Compression block presentation (from Pip Hamilton)
- Description of the firmware structure
  - Description of the inputs and outputs
  - Description of the firmware sub-modules
- Current software interfaces for test and debug
- Implementation results and performance figures
- Current limitations
- Integration with other modules
- To-do list

- **System-level view , memory management**



Compression Overview

Imperial College London

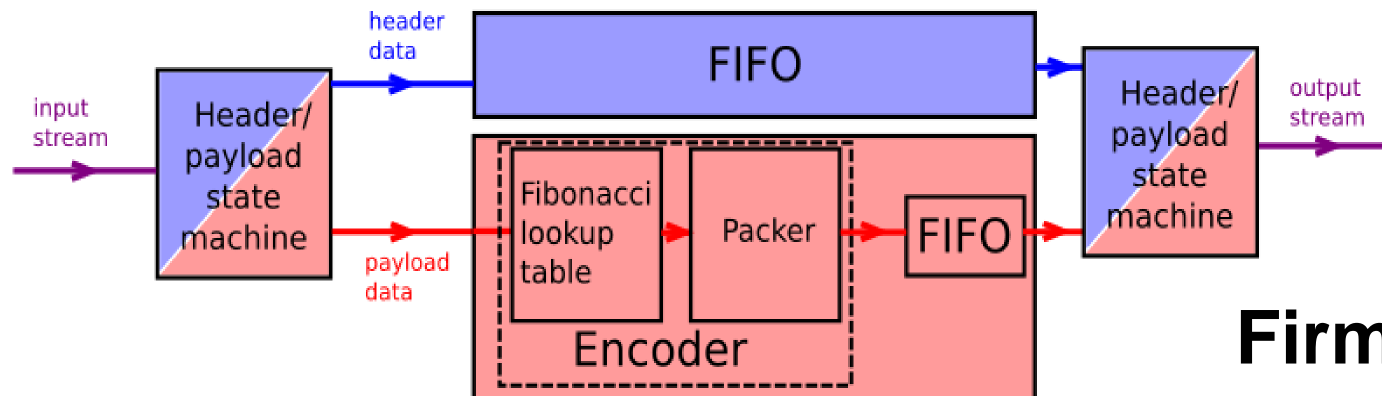
- The compression scheme we are using is Fibonacci encoding, using the Fibonacci sequence as a 'base'.

e.g. 101011 = 1 + 3 + 8 = 12

1 2 3 5 8 ↑

Extra 1 signals end of word:
code words always end uniquely in '11'.

- Encoded number is length i+1, where i = index of largest Fibonacci number smaller than input.
- Should be able to deliver required compression factor C > 2 (likely closer to 3).

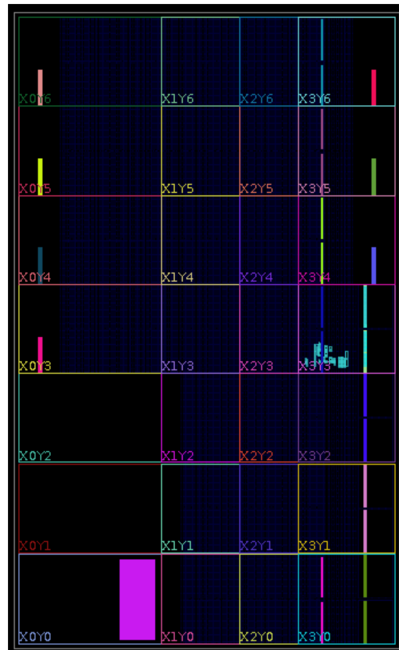


Firmware Implementation

Can be toggled on (default) or off (debug mode, incompressible data)

Performance / Resource Use

- A complete v.0 of the compression module has been written, simulated and synthesised.
- It meets its constraints for timing and resource usage (most demanding requirement being block RAM usage)



- Compression factor delivered currently under assessment.

Timing

Worst Negative Slack	Worst Hold Slack	Worst Pulse Width Slack
1.048 ns	0.016 ns	1.458 ns

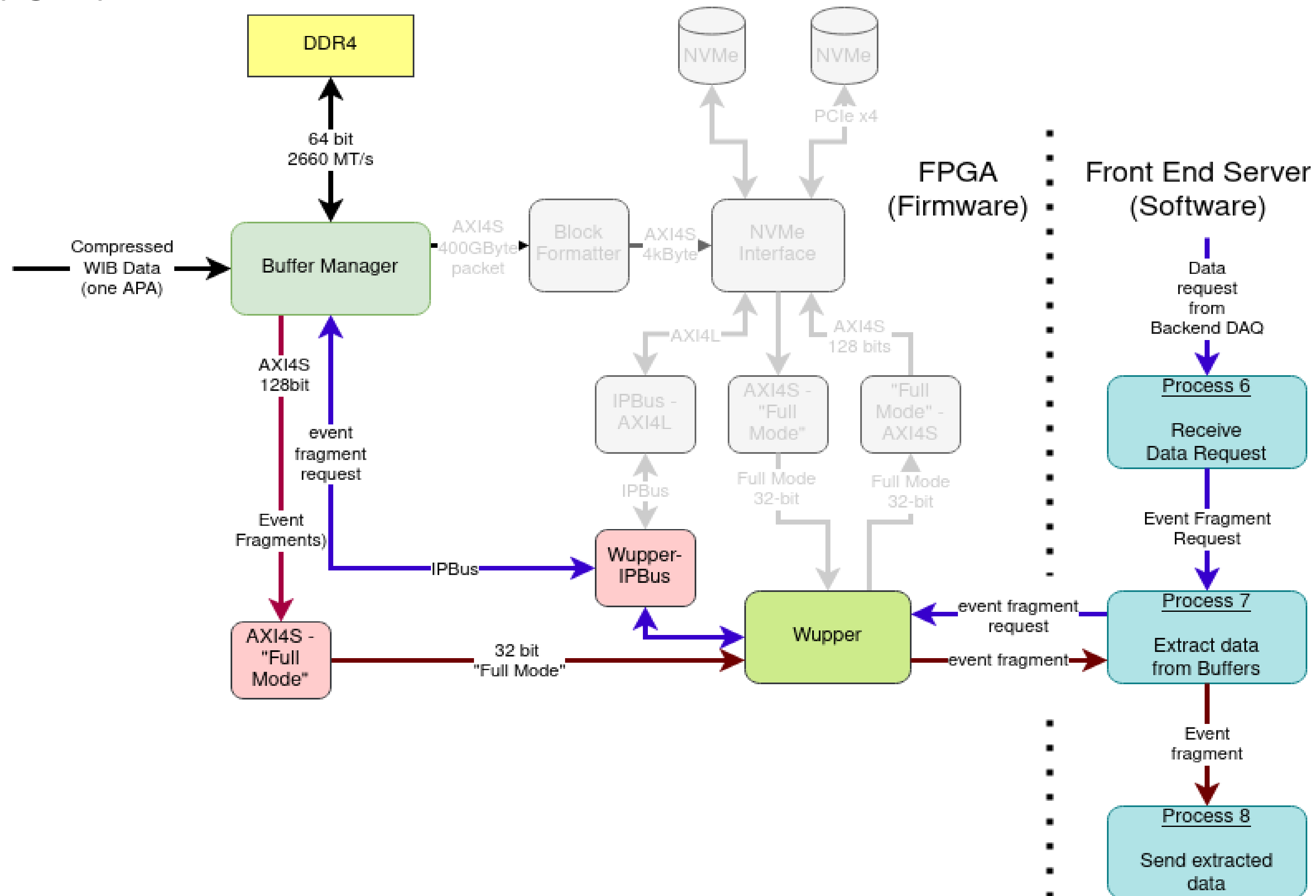
Resource Usage (ZU9EG – ZCU102 board FPGA)

CLB LUTs	CLB Registers	Block RAM
213/274080 **(0.1%)**	337/548160 **(0.1%)**	337/912 **(37%)**

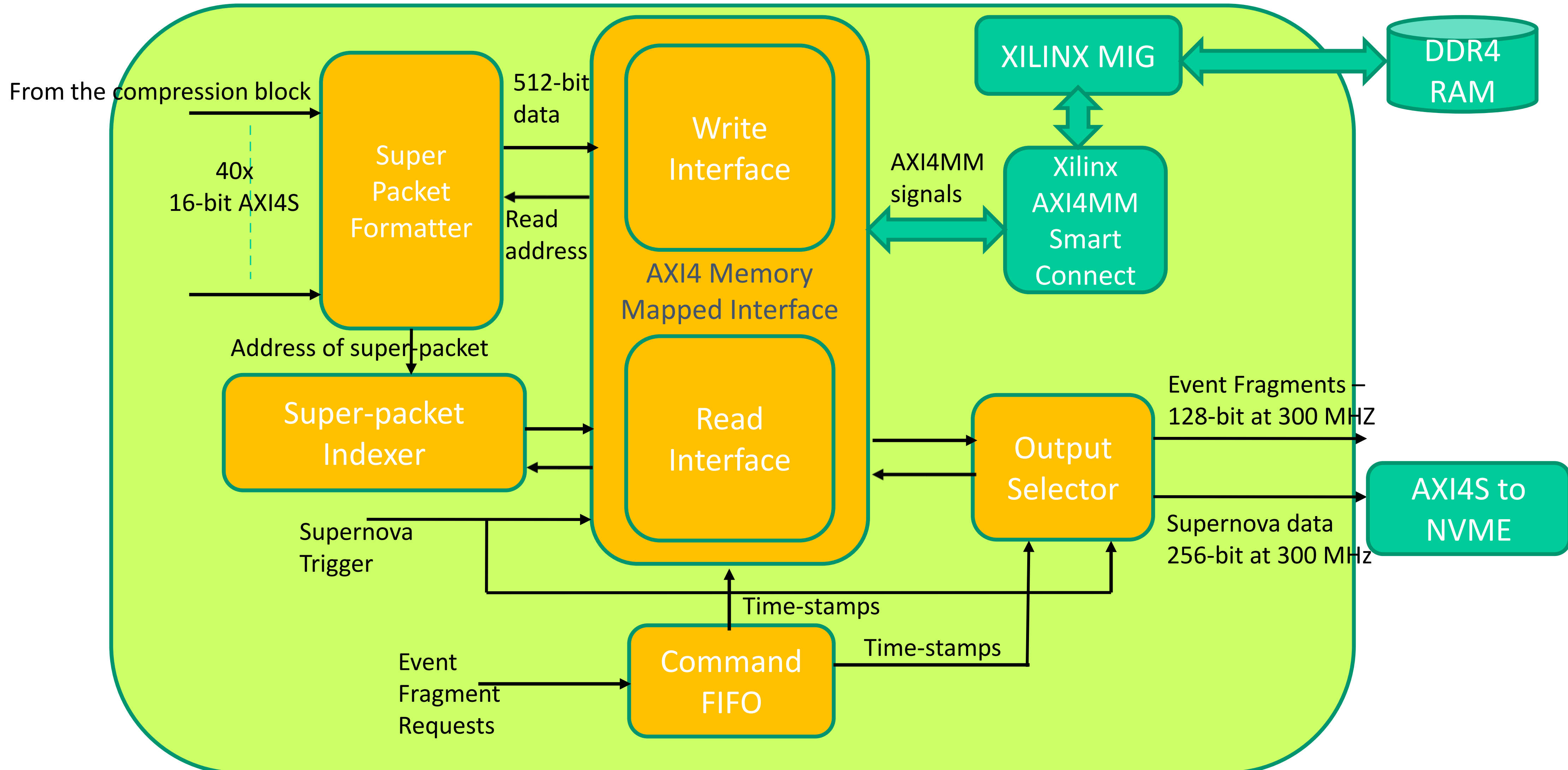
Decompression

- Tools to parse and decode the output of the compression have also been written for validation purposes: these are easily adaptable for use as part of the software chain.
- Initial implementation of on-line decompression routine (Dave Newbold) ~ 80MByte/s per core.

- **System-level view (again)**



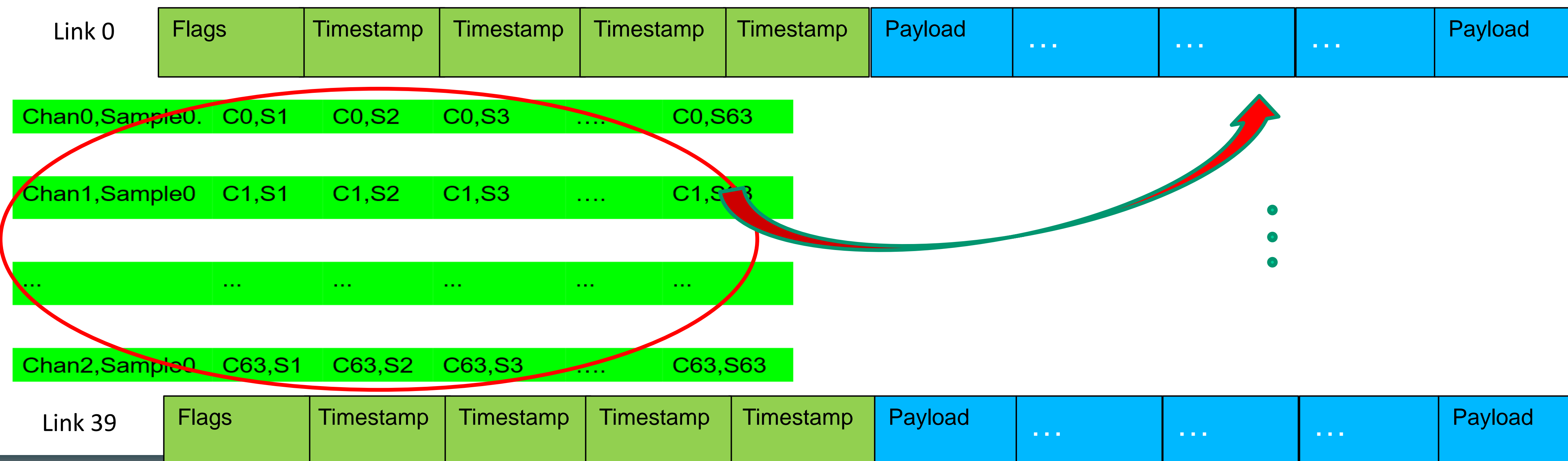
- **10s Buffer management firmware block diagram**



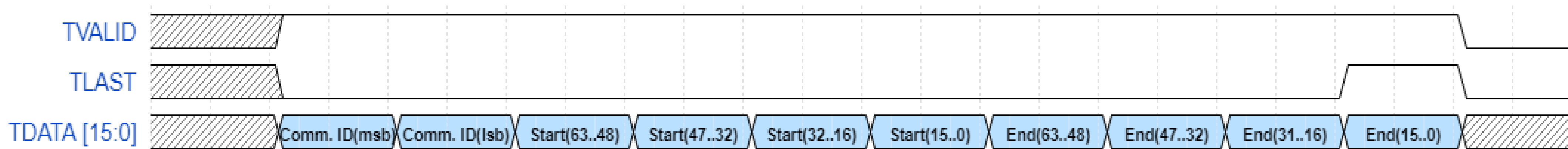
- AXI4 memory interface handles the transition from AXI4 streams to AXI4 memory mapped access to the MIG.

Inputs to the buffer management block – From the compression block

- Compressed "super packets", each carrying 64 12-bit ADC samples from 64 wires, compressed on to a variable number of 16-bit words
- Each super packet comes in a dedicated input link (pipeline) to the block
- 40 of these links corresponding to a total of 2560 wires form the inputs to the block
- The inputs follow the AXI4-stream format that we use for the Upstream-DAQ firmware
- 16-bit words at each clock cycle with extra signals indicating valid data, start and end of the packet, flow-control etc.

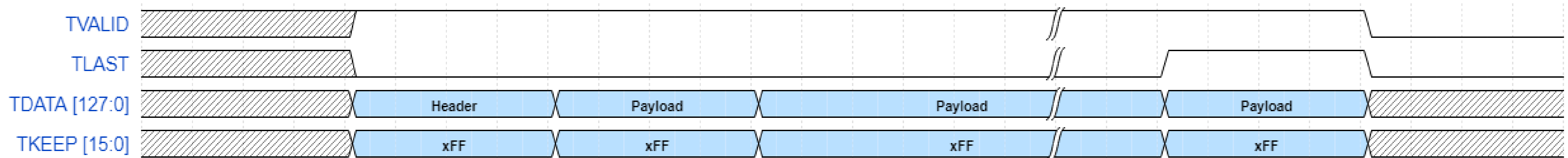


- **Inputs to the buffer management block – From event selection**
  - These are named as event fragment requests in the block diagram – form the "trigger command"
  - Follows the AXI4-stream format with 16-bit words at each clock cycle (250 MHz)
  - Trigger command ID followed by the trigger start time and trigger end time



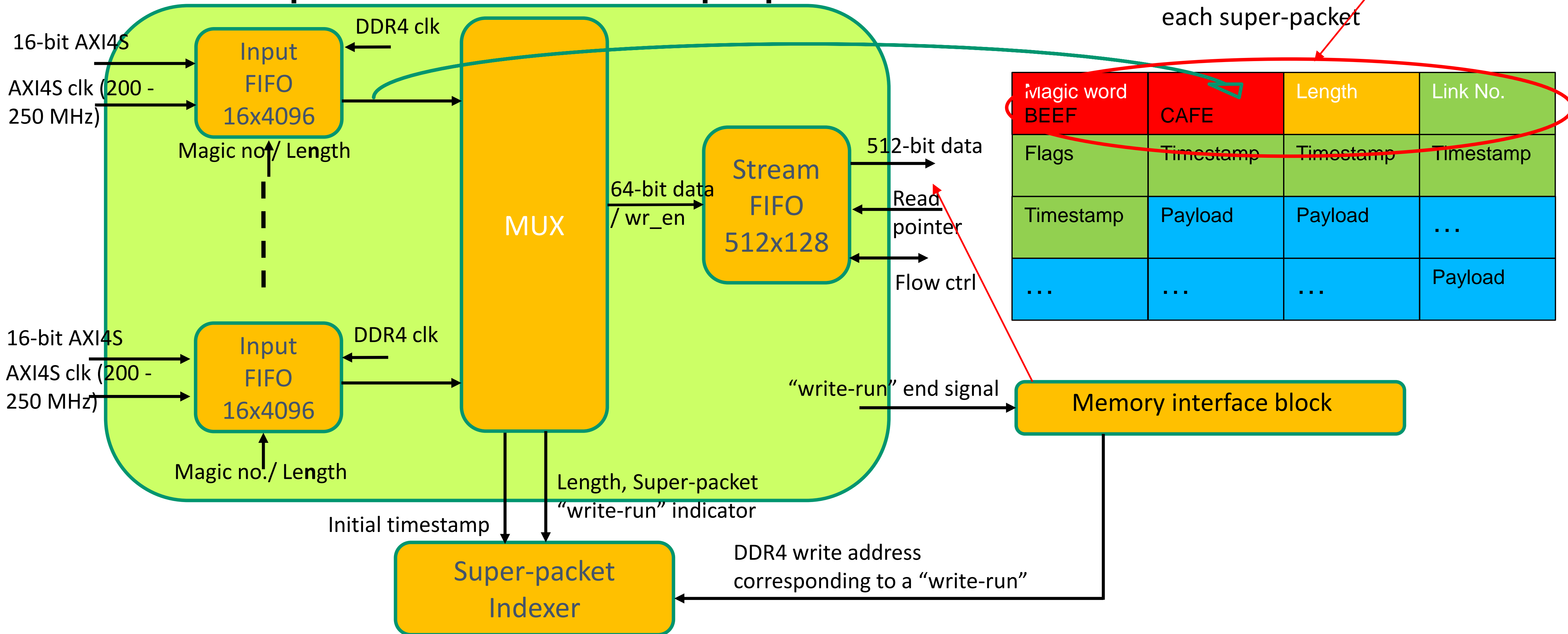
- **Inputs to the buffer management block – From supernova trigger**
  - This starts reading the compressed wire data from the DDR4 memory to the NVME interface
  - NVME interface forms the 100-sec supernova storage buffer
  - A "write" to an IPBus register to start
  - Number of ADC samples to read for the supernova trigger is configurable (at the moment for test purposes – via an IPBus register)

- **Outputs from the buffer management block – Event fragments**
  - These correspond to the data read from the DDR4 RAM as responses to the event fragment requests
  - These follow the AXI4-stream format with 128-bit words at each clock cycle (300 MHz)
  - The command ID of the event fragment request is placed at the start along with the other header information



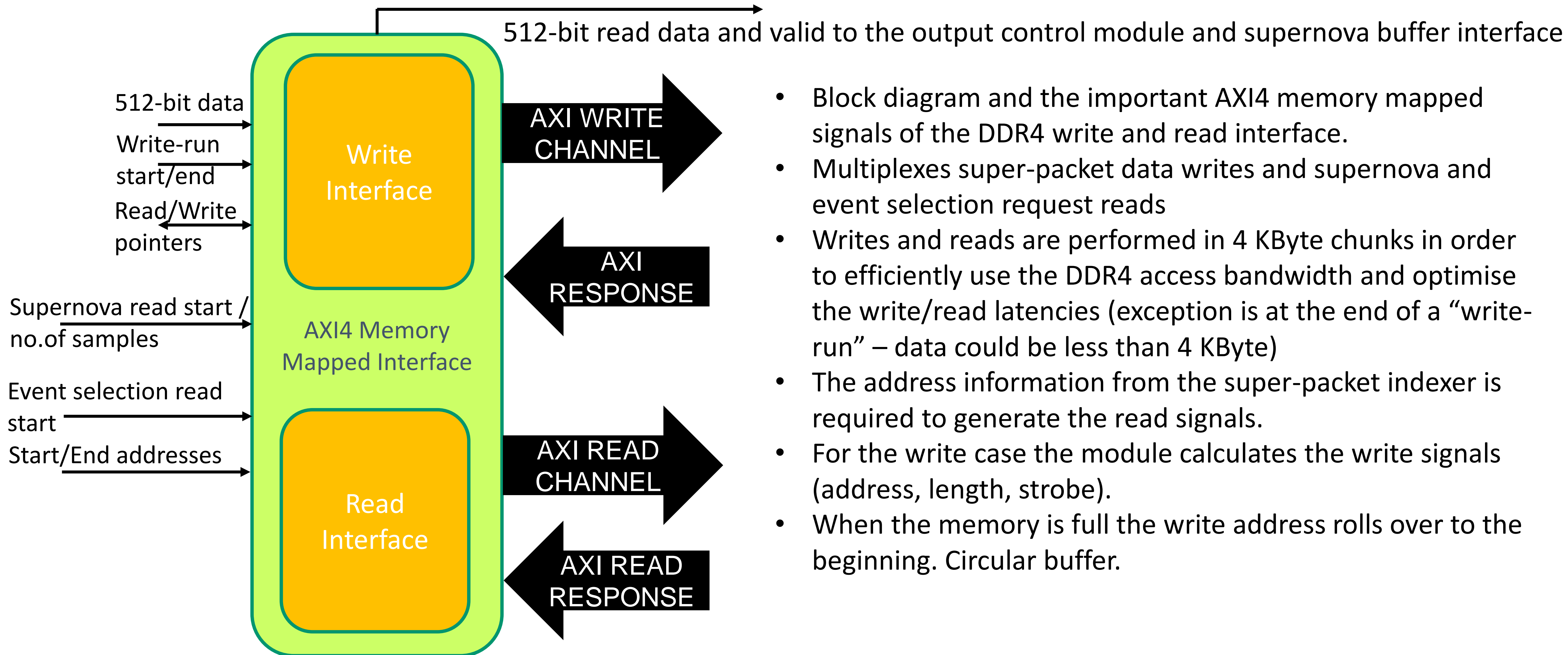
- **Outputs from the buffer management block – Supernova trigger data**
  - These correspond to the data read from the DDR4 RAM as responses to the supernova triggers
  - These follow the AXI4-stream format with 256-bit words at each clock cycle (300 MHz)

- **Detailed description of the firmware – Super-packet formatter**

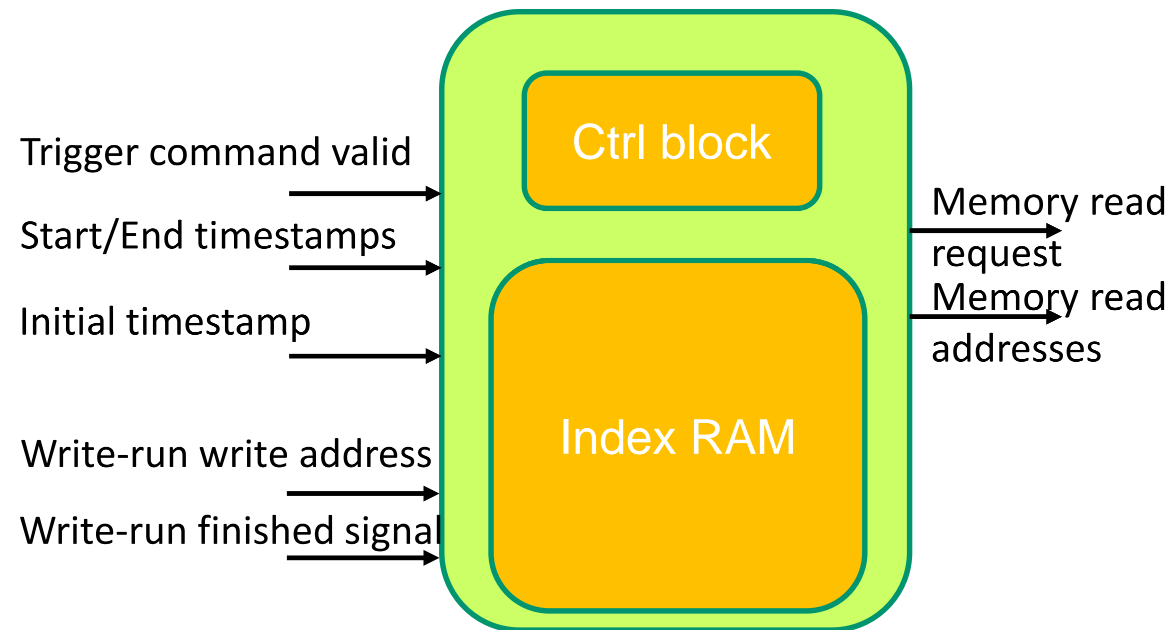


- Mux operates in a round-robin fashion makes sure that all super-packets corresponding to the same time period is written consecutively to the DDR4 memory
- Writing the data from the super-packets from all 40 links corresponding to the same time period is named as a "write-run"

- **Detailed description of the firmware – Memory Interface**

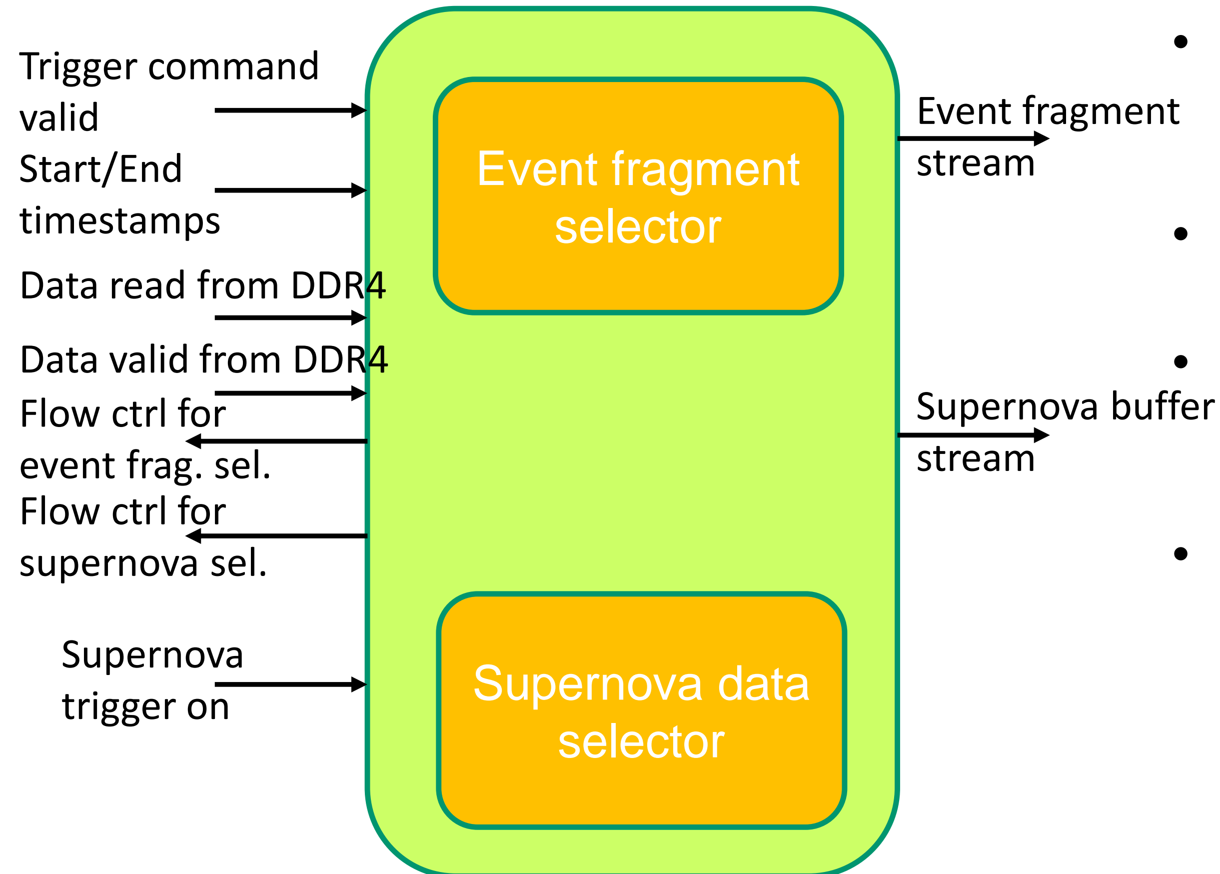


- **Detailed description of the firmware – Super-packet Indexer**



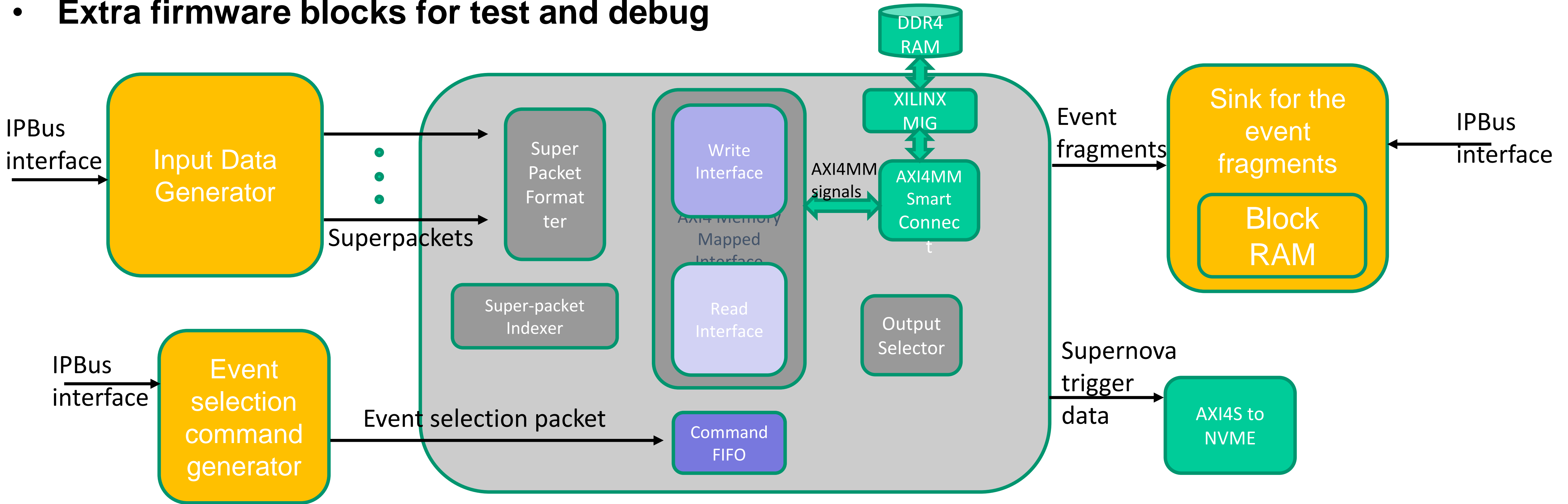
- Super-packet indexer holds the write addresses corresponding to the beginning of each write-run (40 super-packet links)
- Initial timestamp is written at the very beginning of the buffer storage operation
- The current size of the index RAM is 32x16384 – 16K address entries
- Upon an event selection request, the start and end timestamps of a trigger command is sent to the indexer
- Indices of the start and end read addresses are calculated
- These indices are read, and the resulting addresses are sent to the memory interface

- **Detailed description of the firmware – Output selector**



- The event fragment output selector detects the magic word from the incoming DDR4 data to detect a super-packet
- The timestamp is compared against the start and end times from the current event selection command
- Each sub-block has their own FIFOs for the incoming data – flow ctrl signals are generated to provide backpressure
- As a result in the extreme cases loss of data can occur – old data is written over without being read

- **Extra firmware blocks for test and debug**



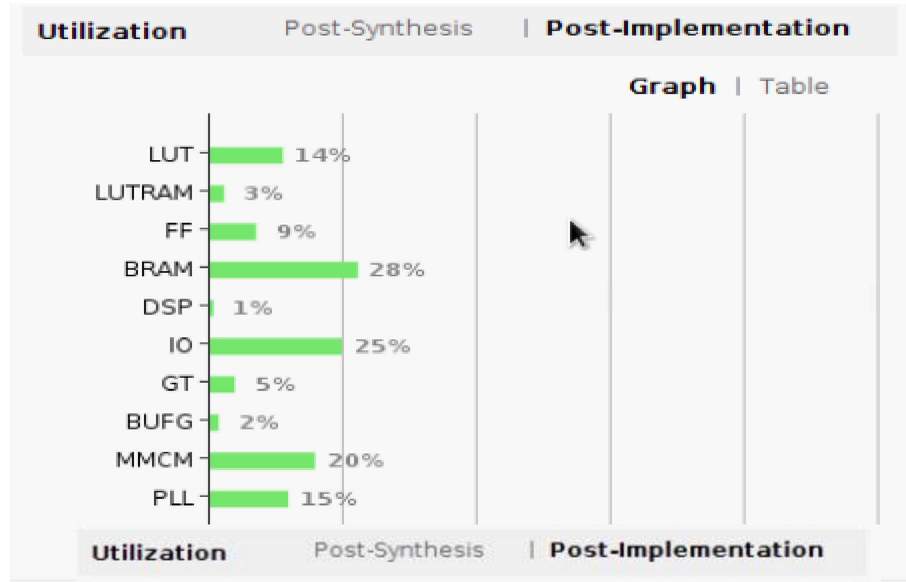
- The input data generator block generates the super-packets from an initial timestamp.
- For each super-packet run the timestamp is incremented by 64
- The data is a counter counting from 0 to payload length – For verifying correct write/read operation
- Event selection command generator generates an event selection packet from IPBus writes
- The sink takes a snapshot of the output in a RAM to be read by IPBus

- Software Interfaces
- Current IPBus registers for debug:
  - **Initialise data send** : Starts sending the data to be written to the DDR4 memory. The data is generated in the raw data packet format and has a counter for ADC samples. 40 channels of data exist each corresponding to a super packet.
  - **Number of packets to be sent** : This sets the number of super packets to be sent to the buffer management block.
  - **Event selection command generate** : These registers are used to generate the event selection command. There is a register for command ID, trigger start time and trigger end time.
  - **Event selection command issue :** This starts sending the event selection command generated.
  - **Input FIFOs enable :** This enables the FIFOs at the input of the buffer management which means enabling the whole buffer management operation.
  - **Supernova trigger :** This starts the reading of the data stored in DDR4 for the supernova trigger.
  - **Supernova trigger no of samples :** This sets the number of samples (no of 16-bit words) to be read for the supernova trigger.
  - **B128 sink :** This is a snapshot FIFO storing the data read from the DDR4 memory as a result of an event selection command.

- **Implementation results**

- The buffer management block and the corresponding test/debug blocks are implemented for two different hardware platforms: ZCU102 board and the KCU105 board – timing closure is achieved for both

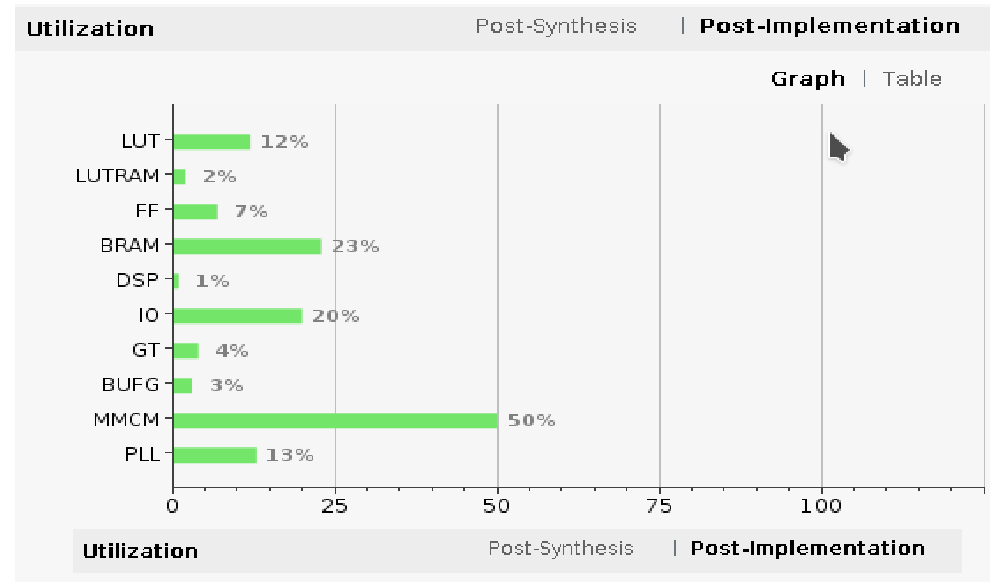
KCU105



Graph | Table

Resource	Utilization	Available	Utilization %
LUT	33197	242400	13.70
LUTRAM	3383	112800	3.00
FF	42077	484800	8.68
BRAM	170	600	28.33
DSP	3	1920	0.16
IO	128	520	24.62
GT	1	20	5.00
BUFG	10	480	2.08
MMCM	2	10	20.00
PLL	3	20	15.00

ZCU102



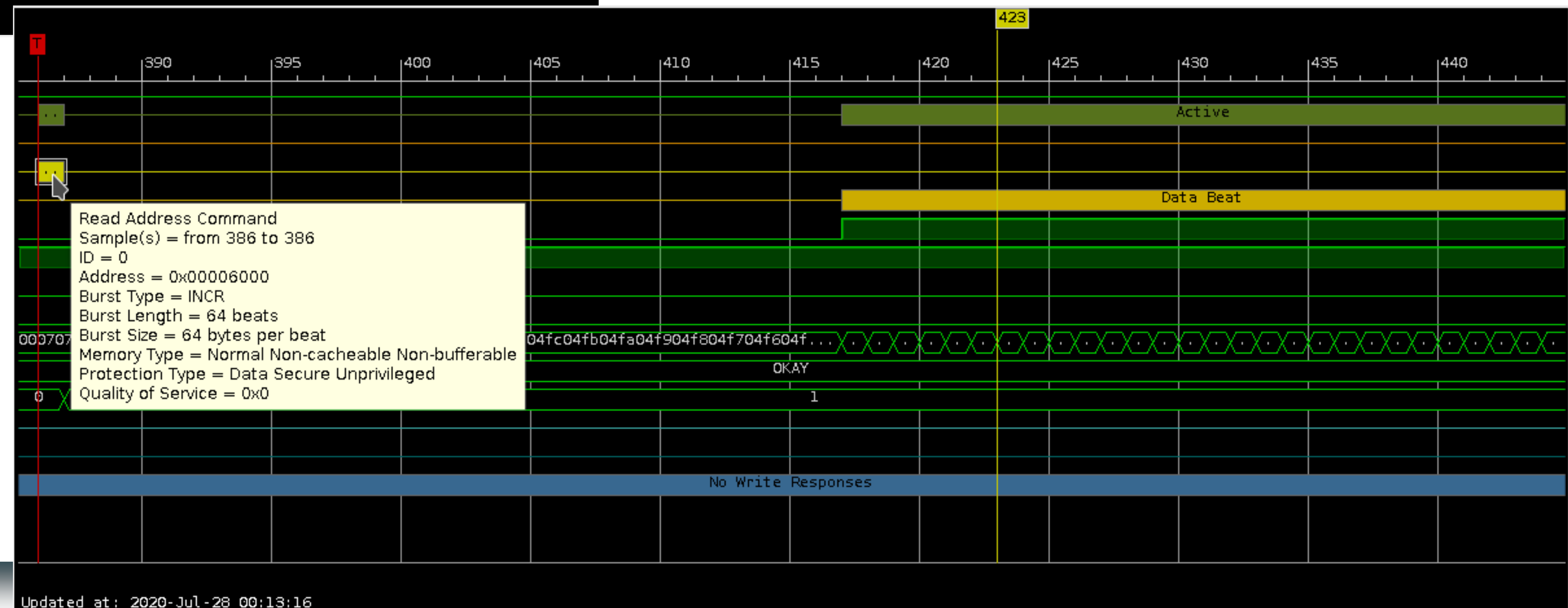
Graph | Table

Resource	Utilization	Available	Utilization %
LUT	32017	274080	11.68
LUTRAM	3449	144000	2.40
FF	40043	548160	7.30
BRAM	205.50	912	22.53
DSP	3	2520	0.12
IO	67	328	20.43
GT	1	24	4.17
BUFG	11	404	2.72
MMCM	2	4	50.00
PLL	1	8	12.50

- **Implementation results** – Visualisation of a write and read operation (4KB size)

Write Operation

Read Operation



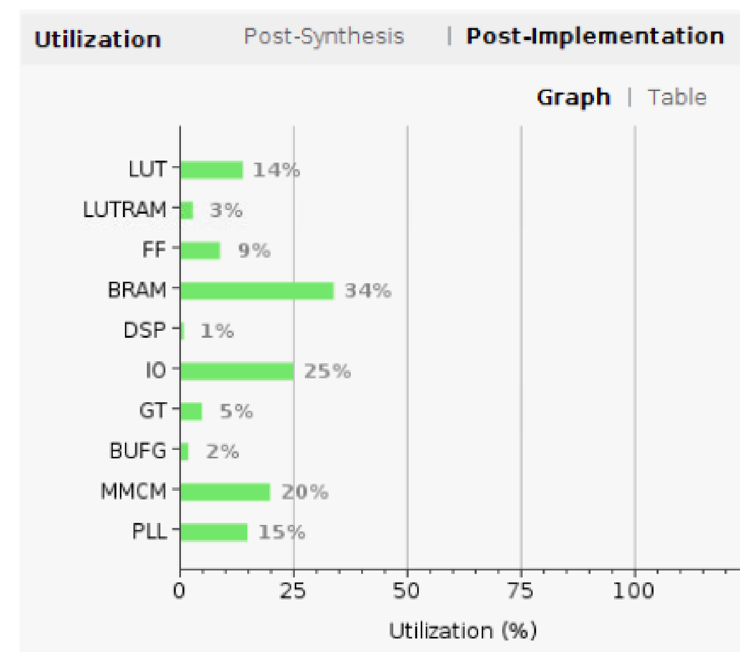
UCL

- **Implementation results – Latencies and throughput for KCU105**
- 64-bit physical RAM connection for KCU105 (faster), 16-bit physical RAM connection for ZCU102
- Write and read throughputs are 512-bit at 300 MHz (19.2 GByte/s – theoretical maximum AXI4 access through MIG) - in practice this figure is lower (latencies and other effects on throughput)
- Using 4KByte accesses optimize the latencies and throughput
- In practice (in the case of constant writes and reads) average write latency ~15 clock cyles (50 ns)
- In practice a 4KB write operation takes ~83 clock cyles (276.4 ns)
- Achievable write speed 14.48 GByte/s
- In practice average read latency ~30 clock cycles (100 ns)
- In practice a 4KB read operation takes ~98 clock cycles (326 ns)
- Achievable read speed 12.26 GByte/s
- The proof of concept design shows the memory access speed is adequate for the application – 2MHz x 2560 x 2 = 10.24 GByte/s incoming data speed – With ~2.6 compression it’s ~4 Gbyte/s

- **Current limitations**
- The size of the super-packet indexer is the main limitation
- Currently 16384 entries are held which correspond to 32us x16K
- The current data selection granularity is 32 us
- This can be increased to support longer time periods
- The size of the index RAM can be increased as well – More BlockRAM usage

- # Integration with the nVME interface
  - The buffer management block is integrated with the NVMe stream formatter for the KCU105 board – timing closure is achieved with the stream formatted running at 300 MHz.

Name	CLB LUTs (242400)	CLB Registers (484800)	CARRY8 (30300)	F7 Muxes (121200)	F8 Muxes (60600)	CLB (30300)	LUT as Logic (242400)	LUT as Memory (112800)	Block RAM Tile (600)	DSPs (1920)	B
top	34674	45247	255	733	161	7285	31096	3578	45247	201	
buf_man (buf_man)	29800	37402	195	682	147	6045	26525	3275	161.5	3	
block_design_inst (design_block	20328	27552	74	355	0	3945	17093	3235	25.5	3	
csr (ipbus_syncreg_v)	59	133	0	0	0	37	59	0	0	0	
csr_en_input_fifos (ipbus_syncre	28	51	0	0	0	24	28	0	0	0	
csr_init_send (ipbus_syncreg_v_	28	51	0	0	0	26	28	0	0	0	
csr_trig_send (ipbus_syncreg_v_	27	51	0	0	0	25	27	0	0	0	
fabric (ipbus_fabric_sel__parame	68	0	0	0	0	31	68	0	0	0	
indexer_trig_com_fifo (super_pa	267	478	10	0	0	107	267	0	16.5	0	
input_fifos (stream_input_block)	4137	4263	4	327	147	1276	4097	40	87.5	0	
ipbus_sink (b128_sink)	259	332	12	0	0	73	259	0	16	0	
output_ctrl (output_control)	1972	795	70	0	0	334	1972	0	8	0	
snv_output_ctrl (output_control_	125	924	7	0	0	172	125	0	8	0	
source_inst (sources)	2347	2344	18	0	0	629	2347	0	0	0	
trig_com_gn (trig_command_ger	143	426	0	0	0	50	143	0	0	0	
ctrl (dtpc_ctrl)	19	32	0	0	0	7	19	0	0	0	
dbg_hub (dbg_hub)	476	739	7	0	0	127	444	32	0	0	
fabric (ipbus_fabric_sel)	18	0	0	0	0	8	18	0	0	0	
infra (kcu105_basex_infra)	2992	4461	36	6	1	773	2916	76	16.5	0	
nvme_fmt_0 (nvme_strm_fmt_blk)	528	800	8	0	0	190	528	0	15.5	0	
nvme_smt_debug (ila_0)	841	1813	9	45	13	254	646	195	7.5	0	



Detailed view – Includes a Chipscope block for avoiding logic optimisation

- **To-do list**
- Porting the firmware to other boards containing SSDs
- Testing with the compression and nVME interface
- Better scripting to provide event selection commands at realistic frequencies
- More characterisation of the latencies and throughput
  - Combining writes, supernova reads, and event fragments reads in a stress-test scenario
  - Integration with the compression
  - Integration with the hit finder firmware

23