consorzio nazionale
interuniversitario
per le telecomunicazioni

Packet Manipulator Processor

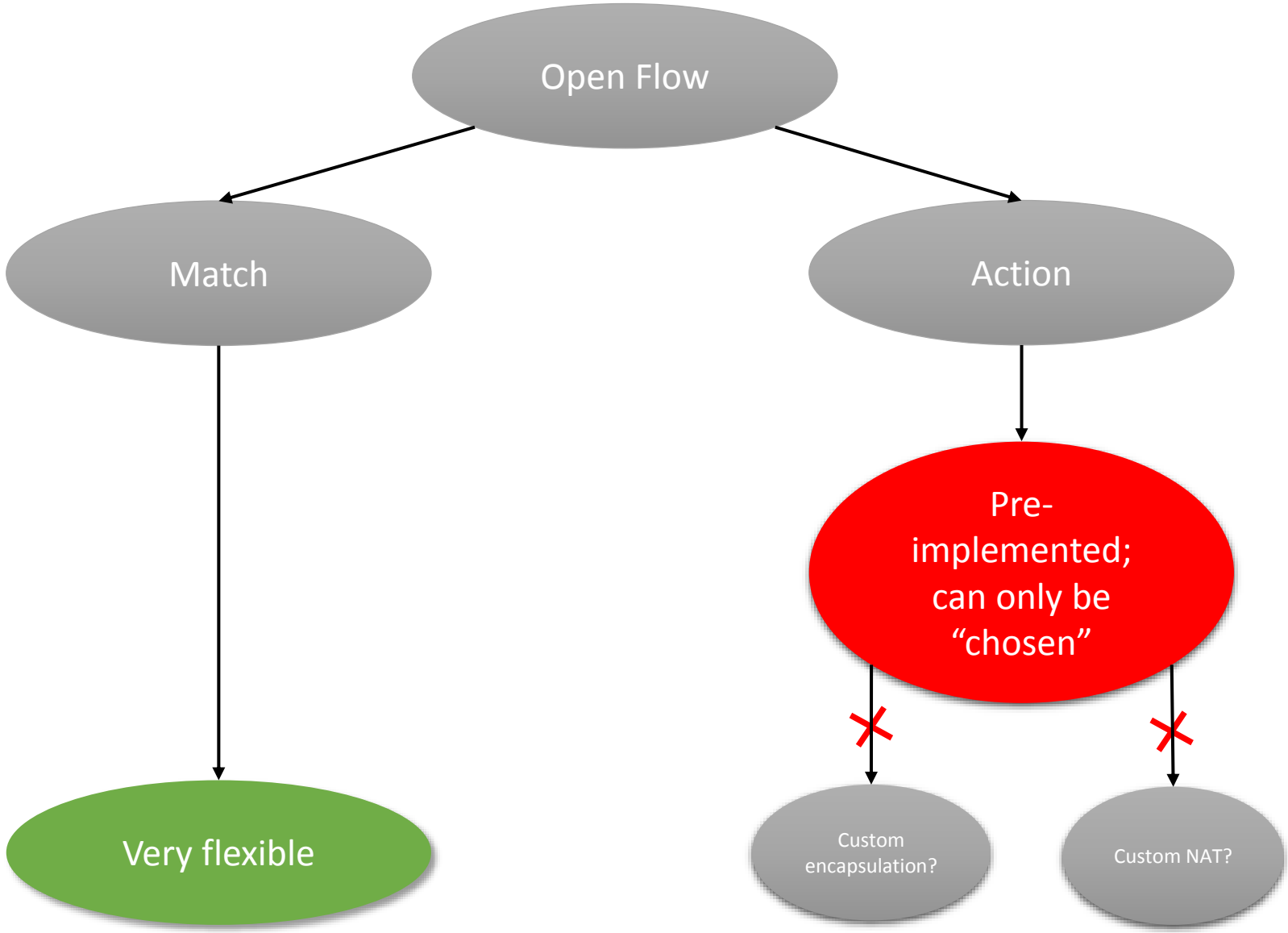
Speaker: Marco Spaziani Brunella

marco.spazianibrunella@students.uniroma2.eu
giuseppe.bianchi@uniroma2.it
salvatore.pontarelli@uniroma2.it
marco.bonola@uniroma2.it

Talk Summary

1. Current state and background
2. What is Packet Manipulator Processor?
3. Programming the PMP
4. Packet Manipulator System
5. Performance highlights
6. Synthesis details
7. Future works, resources and acknowledgements

1– Current state and background



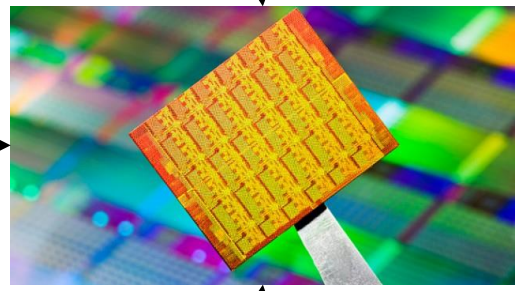
1.1 – Our proposal

Hardware-implemented actions

NAT

IP in IP

ARP reply



Software programmable “actions”

Software code
Eg. ASM, C, P4 …..

Instruction memory

NAT routine

IP in IP routine

ARP Reply routine

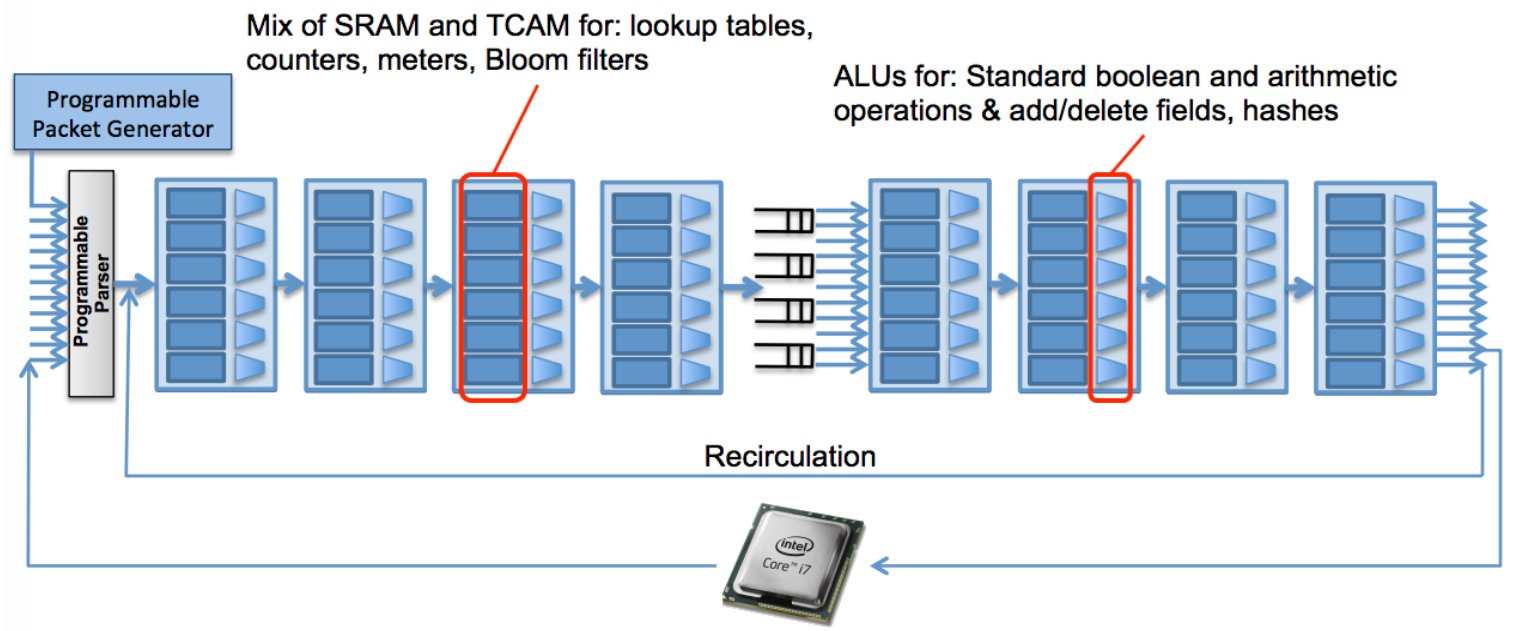
Custom action routine

Packet Manipulator Processor

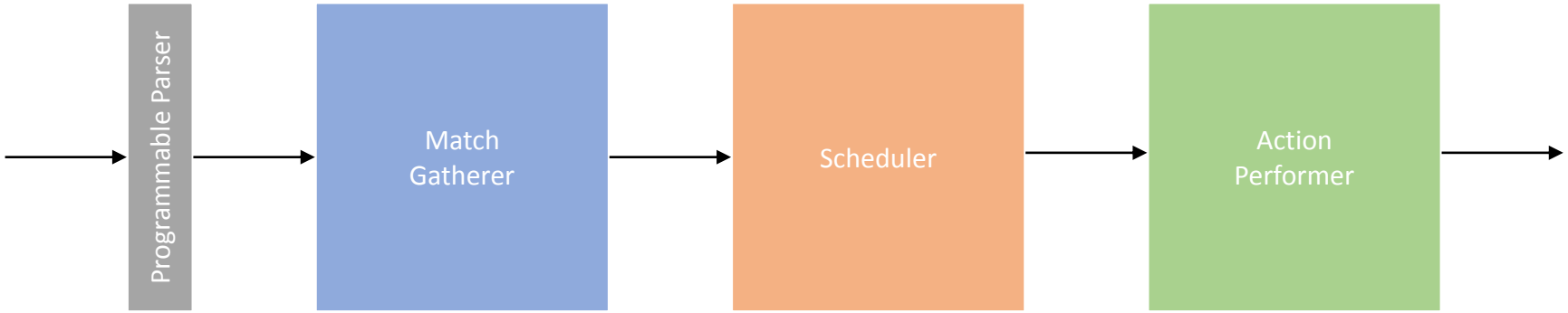
~~Idea 1: MIPS processor~~
Too slow!!!

Idea 2: VLIW processor

1.3 – PISA differences



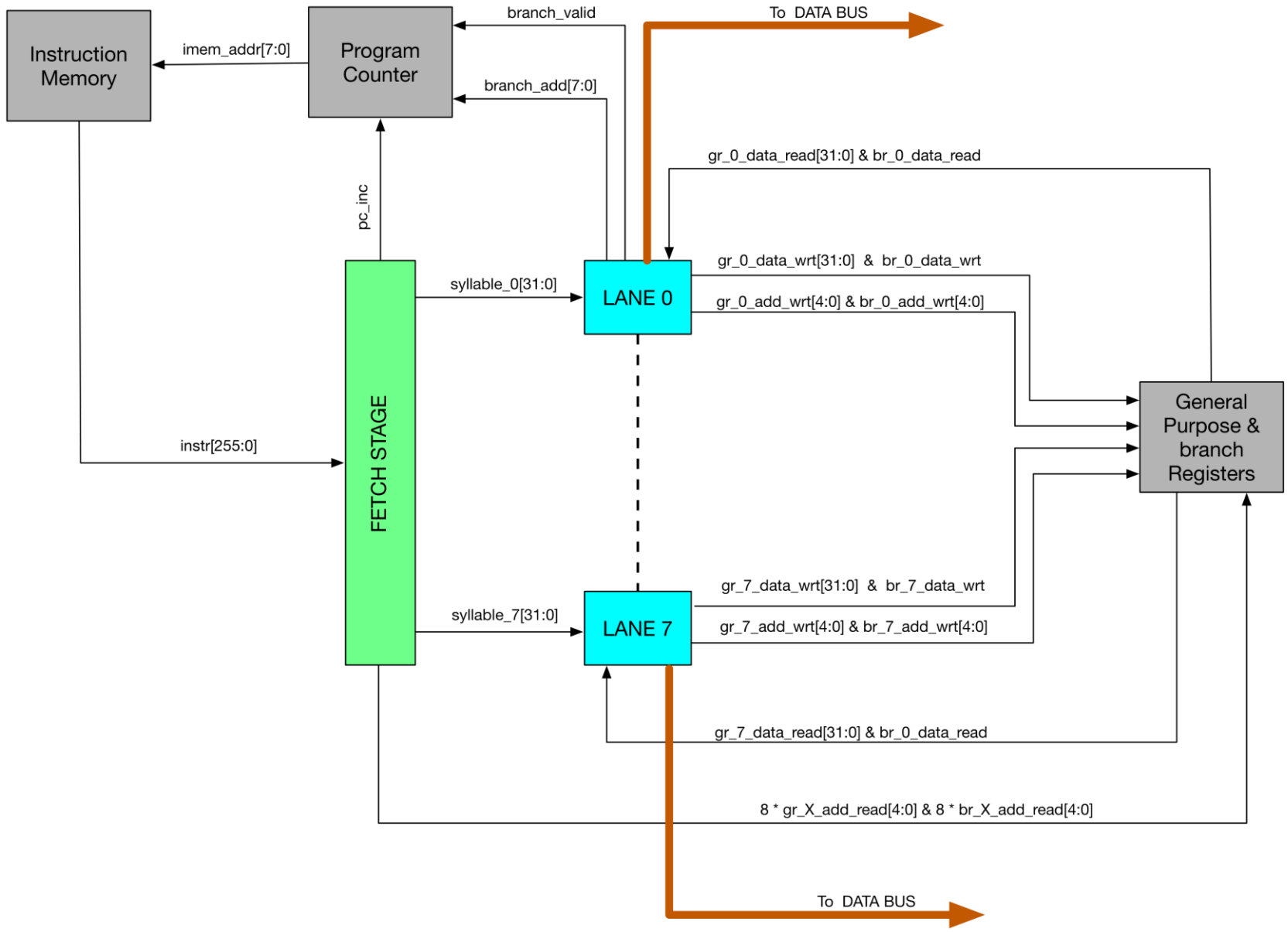
We don't want "atomic" action on a packet going on a pipeline. Instead, we want to collect all the matches for the packet and then arrange the code to perform the desired actions optimizing the hardware.



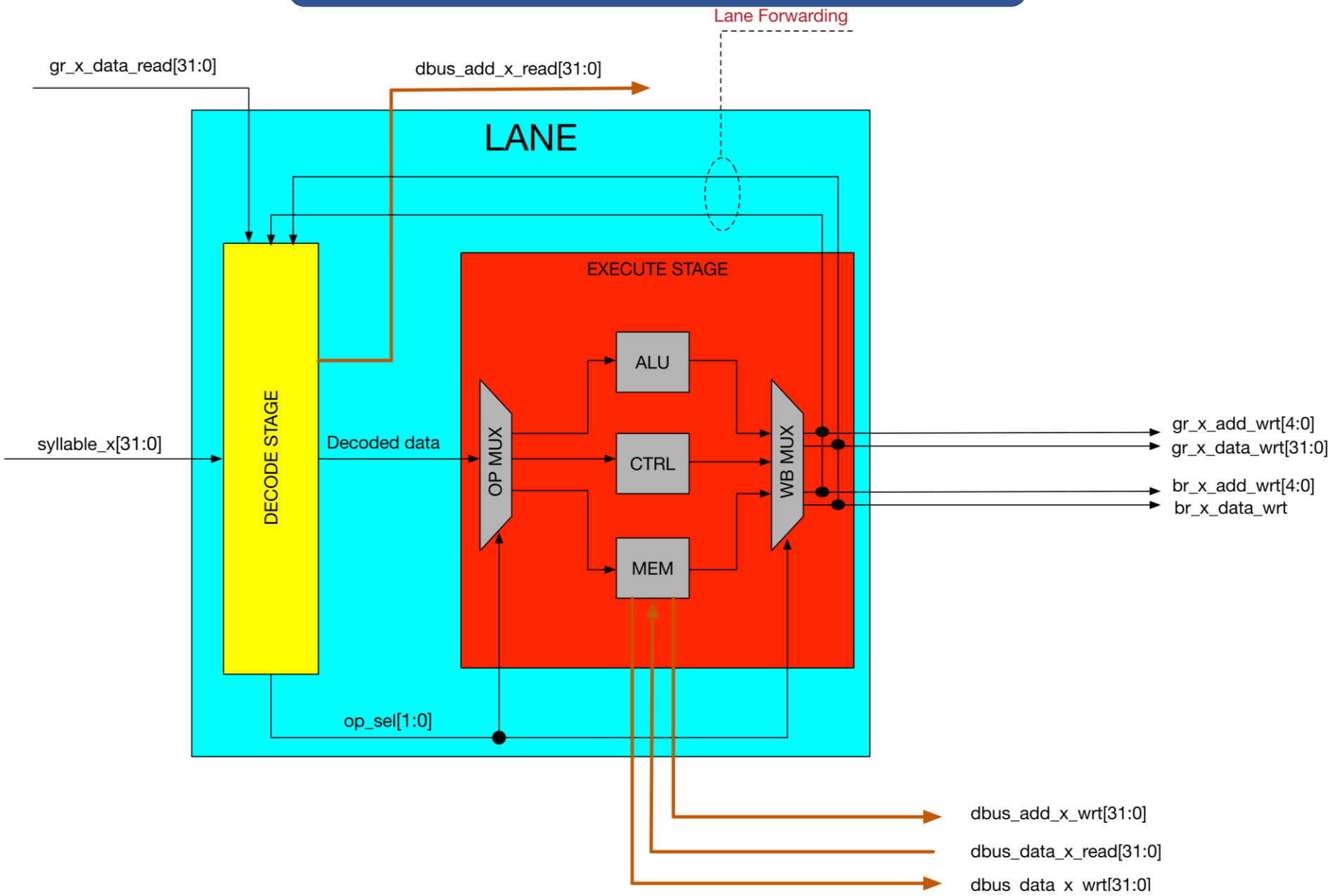
2 – What is Packet Manipulator Processor?

- Static 8-issue microprocessor (VLIW)
- 32bit dataplane
- 32x32bit general purpose registers and 32x1bit branch registers
- VHDL described
- Branch prediction
- Lane forwarding

2.1 – Architectural overview 1/2



2.1 – Architectural overview 2/2



- The instruction set is somehow derived from the one of the MIPS (altough has been enriched with specific instruction to handle branches).

- At the current development phase, the compiler toolchain is missing (we “manually” write the bytecode in the instruction memory)

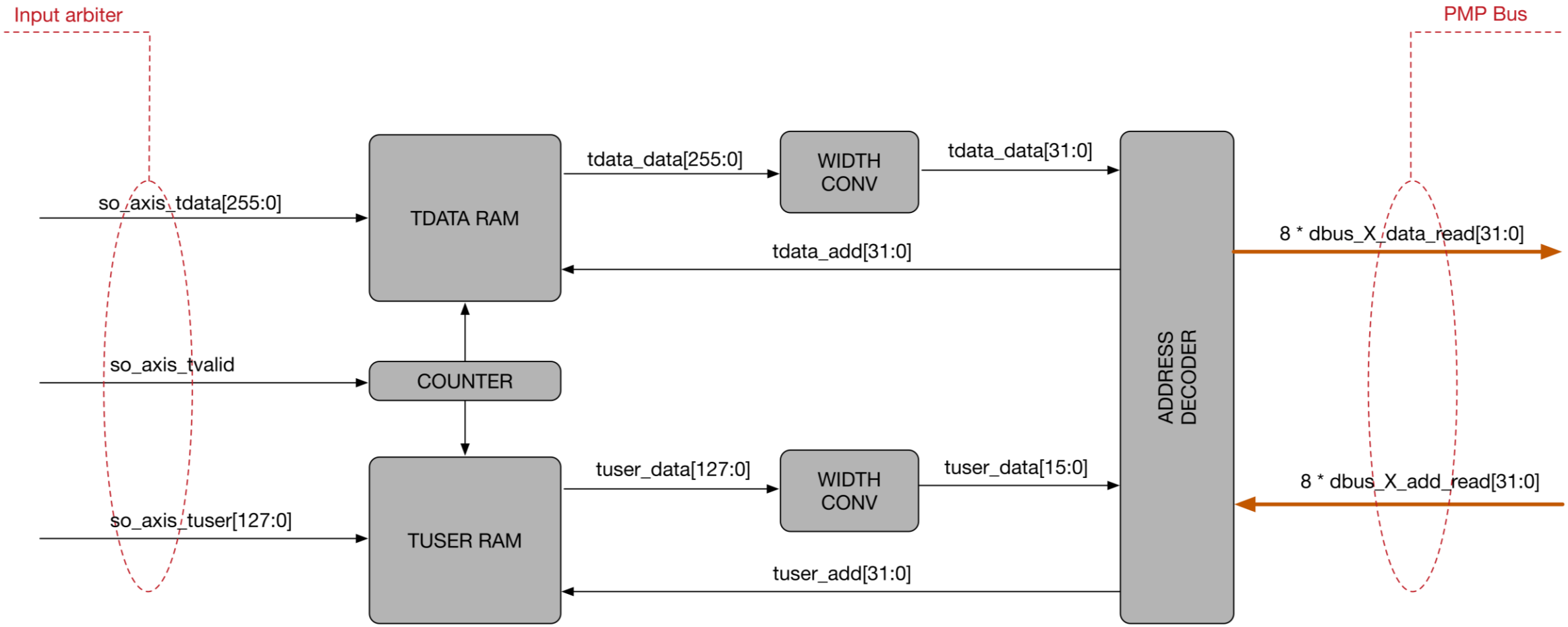
- We are working on an assembler that works with a C compiler for VLIW from HP(not opensource sadly).

4 – Packet Manipulator System

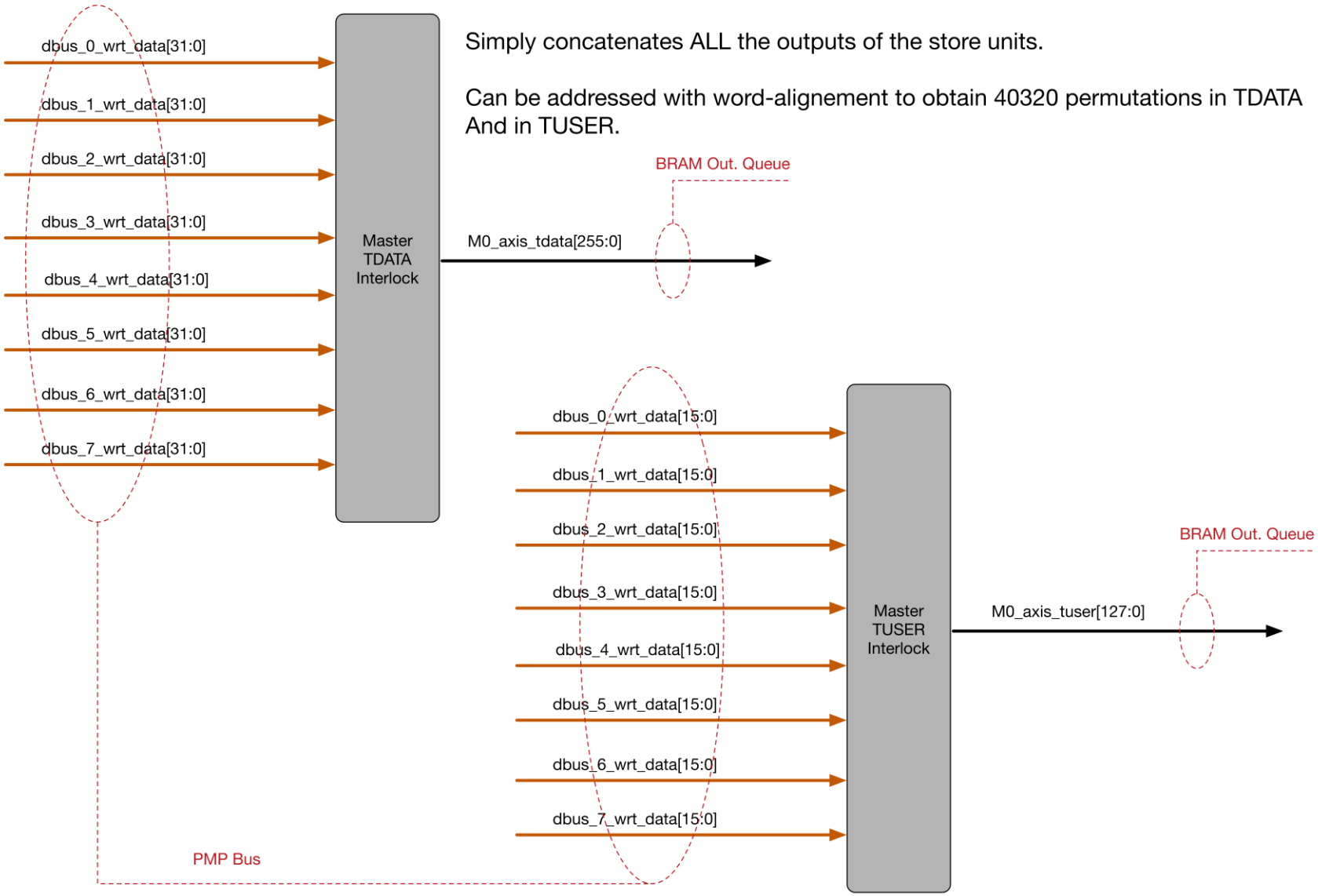
To ease the task of implementing the PMP, we provide a complete PMS (Packet Manipulator System) wich includes:

- PMP Core
- AXI-S Input interface
- AXI-S Output Interface
- Instruction Memory
- Toolchain (Would be nice to have one actually ☺)

4.1 – AXI-S input interface



4.2 – AXI-S output interface



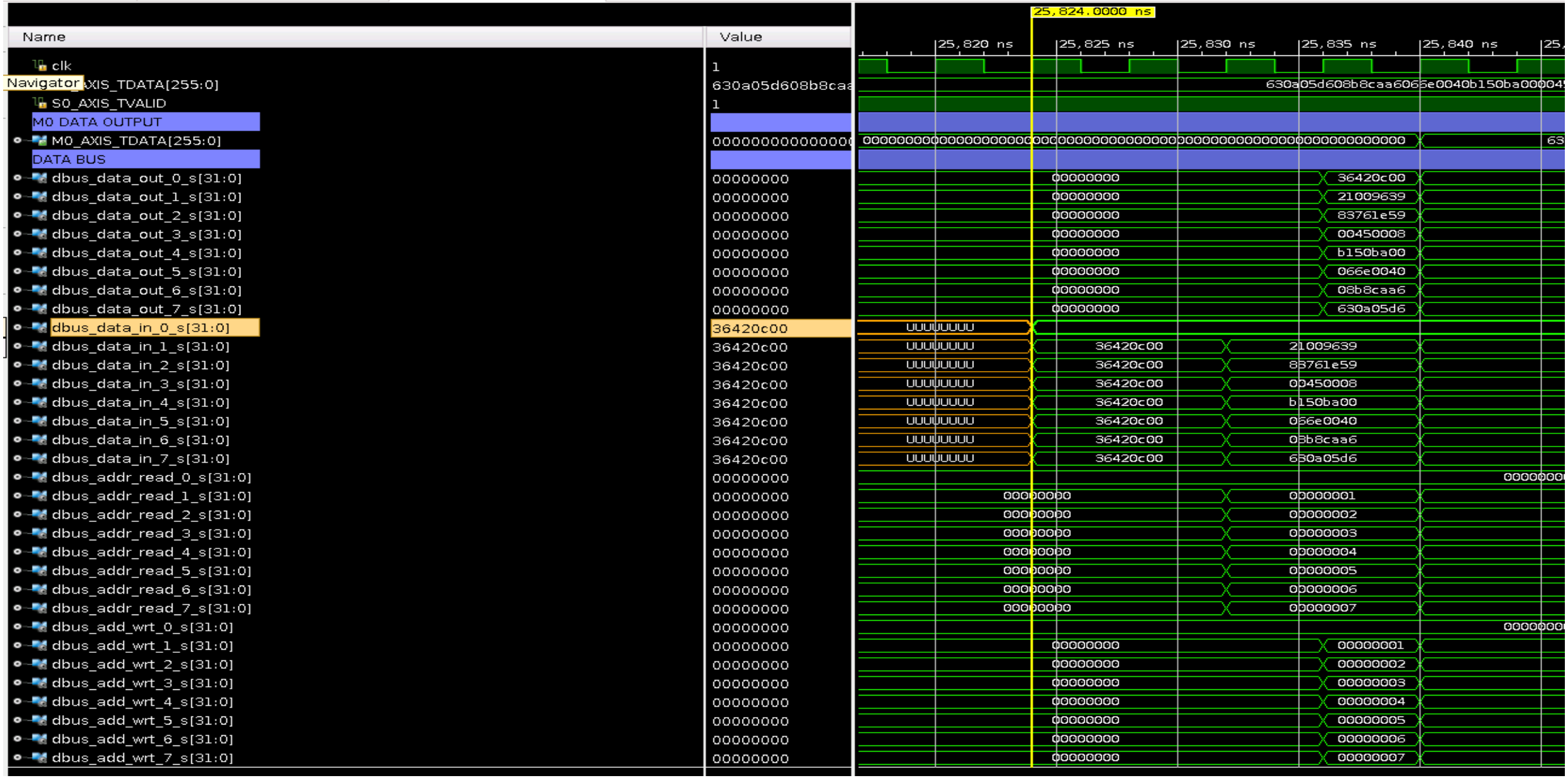
Simply concatenates ALL the outputs of the store units.

Can be addressed with word-alignement to obtain 40320 permutations in TDATA And in TUSER.

5 – Performance highlights

- One instruction (= 8 syllables) per clock cycle
- 3-stage pipeline for each lane
- 3 clock cycle delay
- 1 clock cycle branch penalty
- Heavy use of prefetch (data to be used in decode stage are required in fetch stage…)
- 256 bit parallel load/store in one clock cycle (ideal for AXIS NetFPGA-SUME data interface☺)
- Lane forwarding to avoid penalty arising from the pipes (if data required is the same as the one currently in writeback, the data is forwarded in an earlier stage of the lane to avoid data hazards)

5.1 – Example: load/store from NetFPGA Input arbiter to BRAM queues on 256bits



1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	3224	0	433200	0.74
LUT as Logic	3224	0	433200	0.74
LUT as Memory	0	0	174200	0.00
Slice Registers	1278	0	866400	0.15
Register as Flip Flop	1278	0	866400	0.15
Register as Latch	0	0	866400	0.00
F7 Muxes	112	0	216600	0.05
F8 Muxes	0	0	108300	0.00

2. Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	44	0	1470	2.99
RAMB36/FIFO*	44	0	1470	2.99
RAMB36E1 only	44			
RAMB18	0	0	2940	0.00

Very small HW footprint!

6 – Synthesis details 2/2

Clock Summary

Name	Waveform	Period (ns)	Frequency (MHz)
master	{0.000 2.000}	4.000	250.000

Fully synthesized at 250MHz!
Way over 156.25MHz needed for in-out interface!

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.677 ns	Worst Hold Slack (WHS):	0.020 ns	Worst Pulse Width Slack (WPWS):	1.326 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	27423	Total Number of Endpoints:	27423	Total Number of Endpoints:	7736

All user specified timing constraints are met.

7.1 – Future work

- Provide a complete toolchain for the PMP
- Porting of the ISA to RISC-V
- Adding more network-specific instruction (eg. Checksum recalculation)

7.2 – Resources

- https://bitbucket.org/marco_spaz/pmp
- https://riscv.org/

7.2 – Acknowledgements

This project has been developed within the European project Superfluidity.

Thanks for your attention!