



A two-phased multimedia SoC design optimization using ESL tools

Hoon Oh, Youngkil Park, Byungchul Hong, Chulho Shin and Derek Ko
LG Electronics

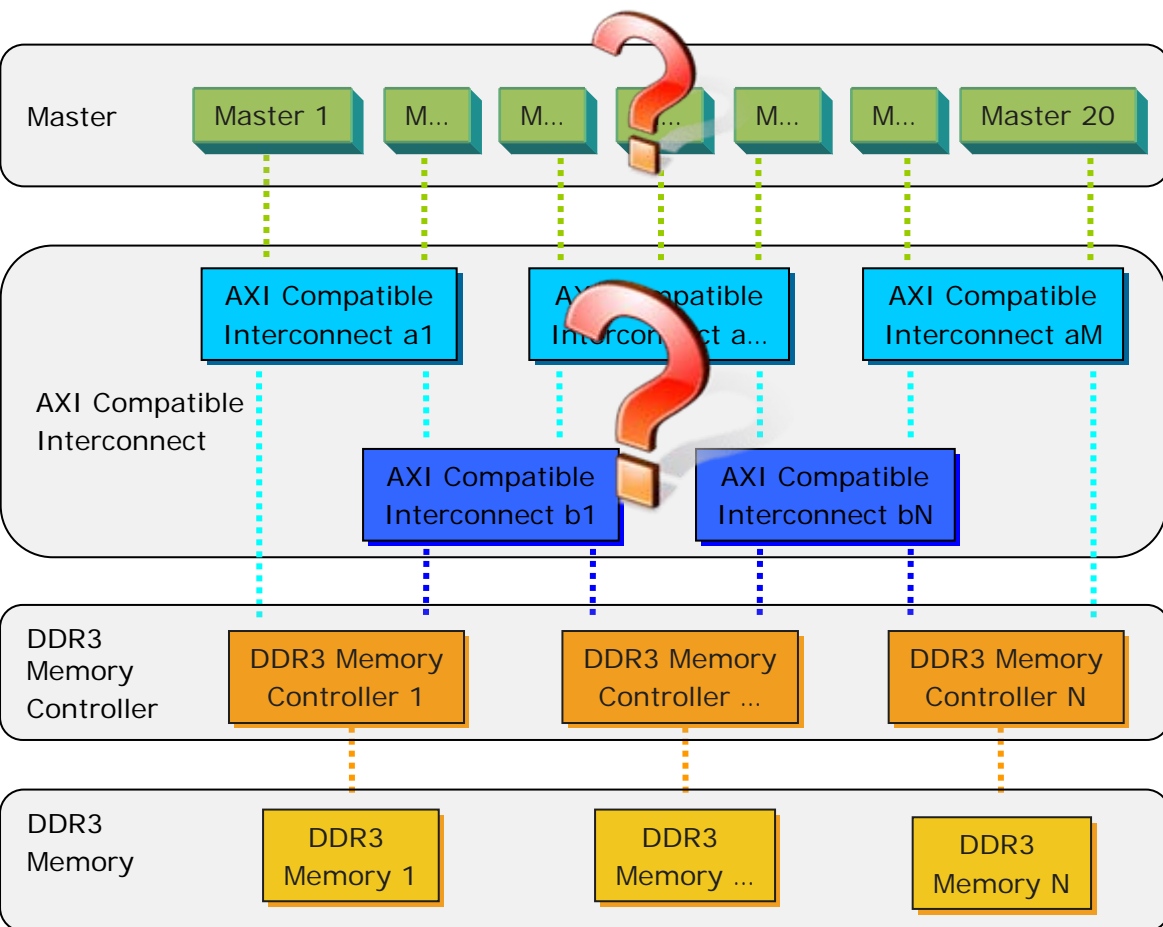
{hoon.oh, youngkil.park, byungchul.hong, chulho.shin and derek.ko}@lge.com



LG Electronics
System IC Business Team



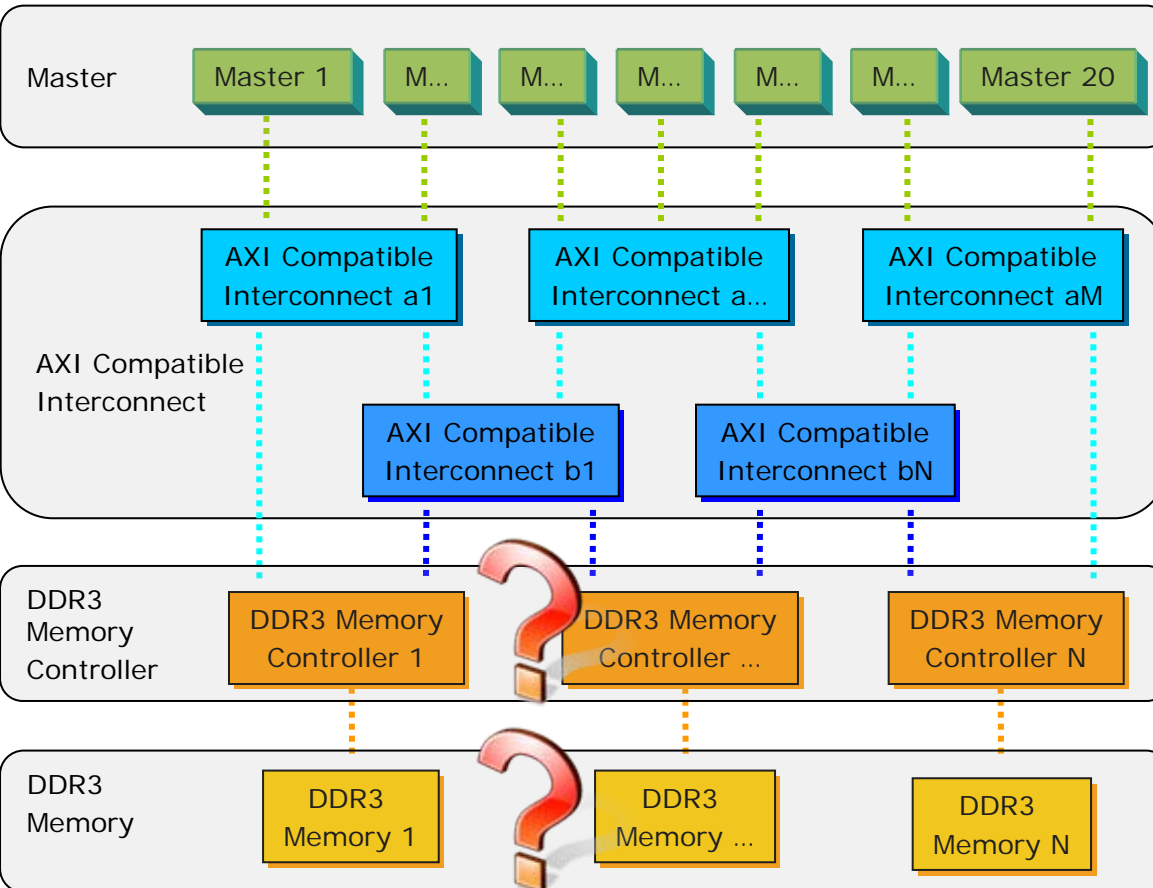
Motivation1 (related to masters and interconnects)



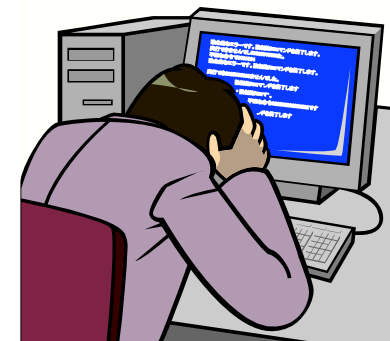
- Many IP blocks under development
- more than 20 masters : memory bandwidth- hungry and latency sensitive multimedia application-specific masters.
- A CPU that runs at 700MHz and GPU for 3D graphics acceleration.
- Visibility & configurability ?

- Optimal architecture for interconnect? (BW, operation frequency, bit-width of the interconnect, gate count, cascade depth)
- What kind of arbitration & QoS scheme needed?

Motivation2 (related to memory controllers)



- Designing DDR3 memory controller : How to estimate its performance according to changes being made.
- Optimal arbitration & QoS scheme?
- External memory BW required by each master?
- Traffic distribution from many masters to multiple memory.





- Limited resources for system architecturing
 - Less than two manpower available except memory controller design
 - No accumulation of ESL library models of each master IP

- chicken-and-egg problem
 - Top system architecturing requires each master's accurate BW and latency information.
 - Each master requires it's available BW and latency constraints
 - What is the precedence? Master IP design or Top architecturing?



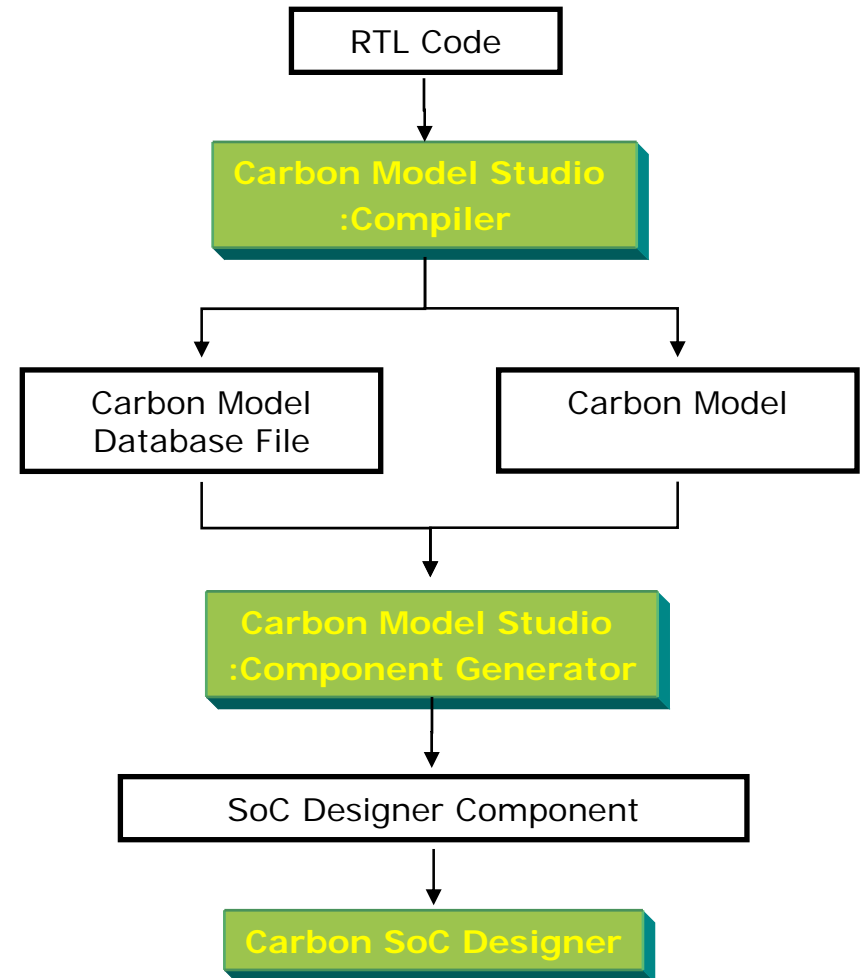
In this talk, we demonstrate a two-phased approach to tackle all these problems.

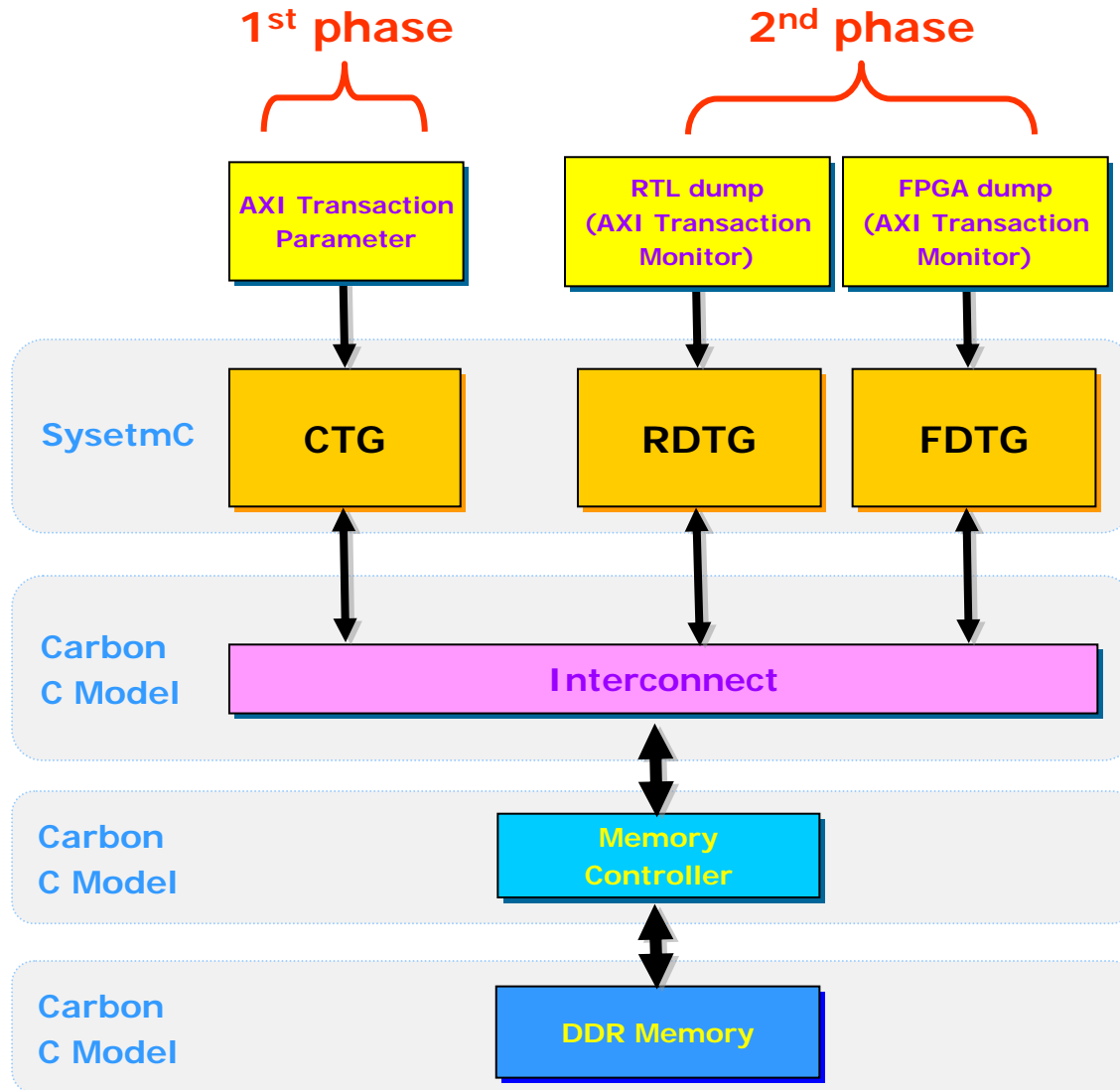
■ Key objectives :

- Maximize external memory BW(bandwidth)
- Keep memory access latency as short as required for latency sensitive masters.

■ For minimal resources (people and time),we decided to use ESL tools:

- Carbon Design System's two major ESL tools to help tackle our problem
 - **Carbon Model Studio** to generate C models from RTL codes
 - **Carbon SoC Designer**
 - to integrate the generated C models for constructing our system
 - to explore various architectures
 - to simulate the system
 - to analyze simulation results





- **CTG (Configurable Traffic Generator.):** Our traffic generator that can behave according to the given set of behavioral **parameters** for memory accesses
- **RDTG (RTL Dump file-based Traffic Generator):** AXI traces were captured from **RTL** simulation
- **FDTG (FPGA Dump file-based Traffic Generator):** AXI traces were captured from **FPGA** emulation

1st phase : AXI master modeling



- We defined many parameters:
 - **AXI protocols** : bit width, interface clock, burst length, transfer size....
 - **Addressing setup** : multiple outstanding, accessing address range, increment/random addressing ratio....
 - **Bandwidth control** : BW quantity, active BW section, latency limit....
 - **Profiling Information** : Profiling on/off, BW profiling, latency profiling, profiling range.
 - There are more parameters....
- The parameters that specifying behavior of each master were **collected from master block designer**.



CTG Component in SoC Designer

VENC0 (AXI_CTG) i

clk-in axi_m

CTG Parameter setting

Edit Parameters - VENC0 (AXI_CTG)

Properties:

Name: VENC0

Type: Other

Version: 1.1

Description:

Debuggers Supported:

Port List:

Port Name	Interface Type
axi_m	TransactionMaster
clk-in	ClockSlave

Parameters:

Parameter	Value	Type
CTG_NAME	VENC0	value
MAX_RID	2	value
RBPS1	53899	value
RBPS2	215501	value
ARSIZE	3	value
R_OUTSTANDING[0][0]	1	string
R_OUTSTANDING[1][0]	1	value
RANDOM_REN[0][0]	0	bool
RANDOM_REN[1][0]	0	value
ARBURST	1	value
ARLEN[0][0]	15	value
PROFILE_EN	0	value

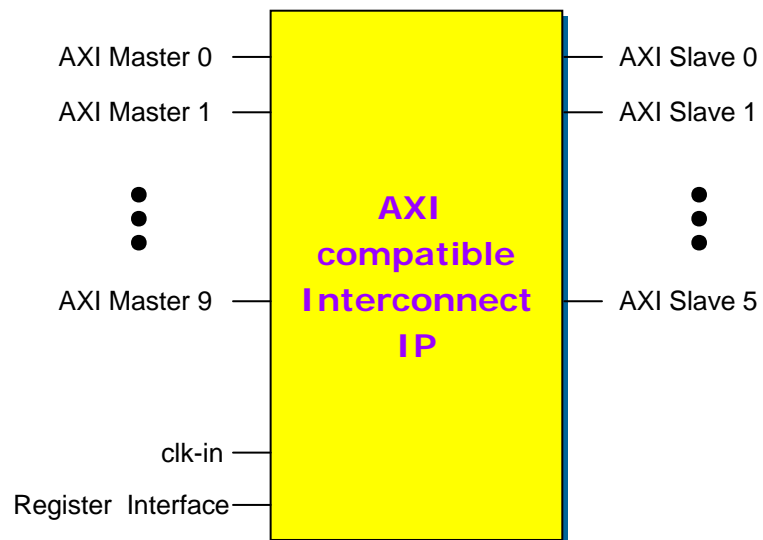
Documentation OK Cancel

+ Refer to Appendix about the parameter table & complex addressing examples.

1st phase : AXI compliant interconnect IP

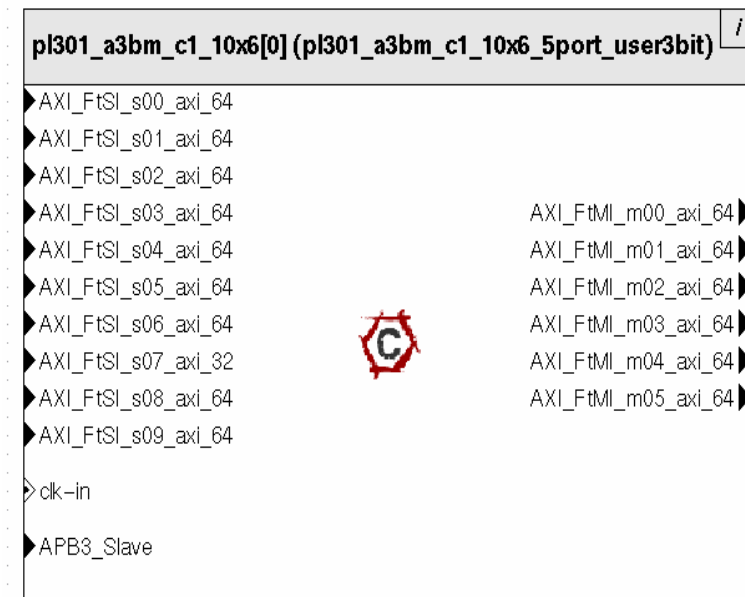


- C model which converted from RTL by carbon model studio.



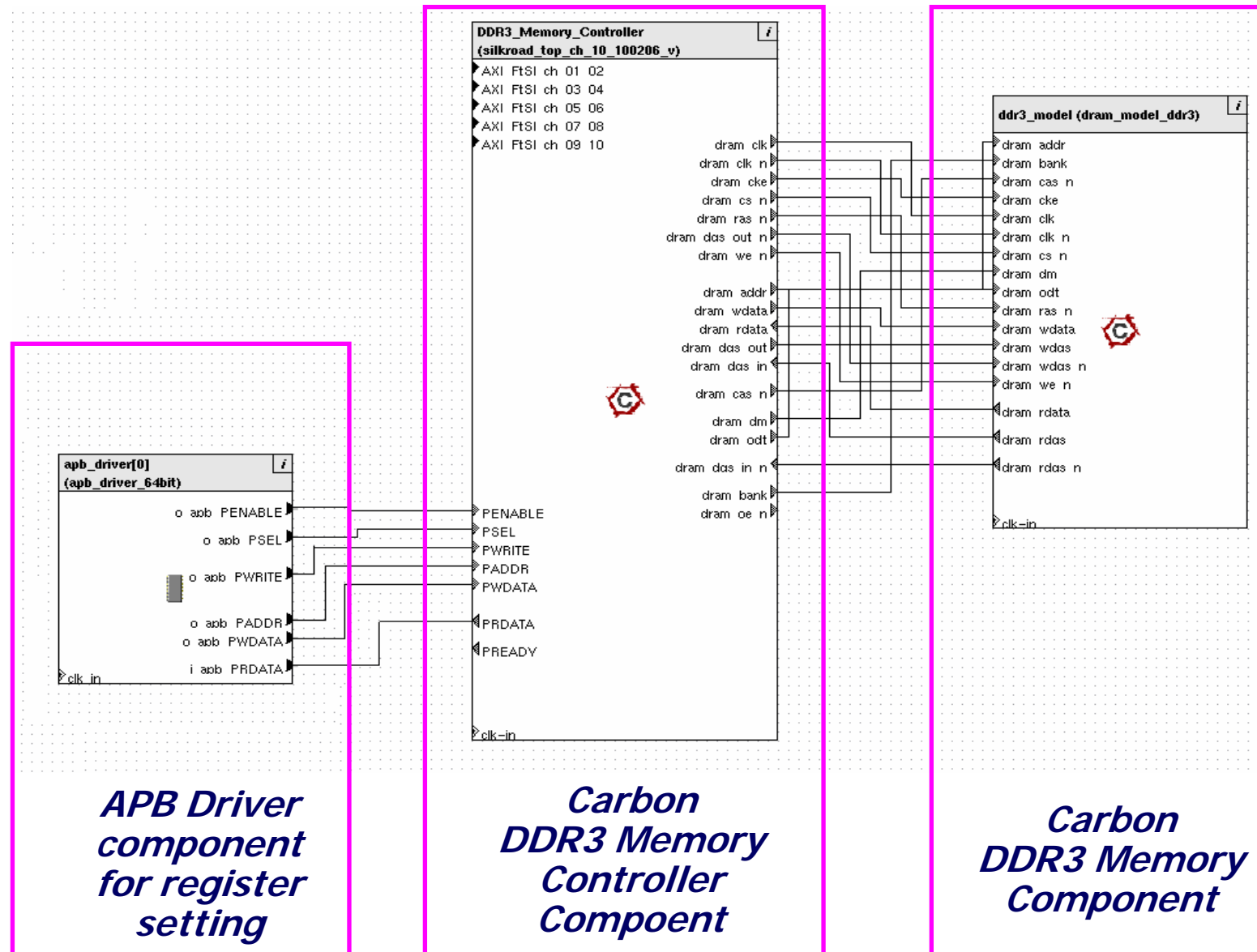
RTL

Create
Carbon
Component

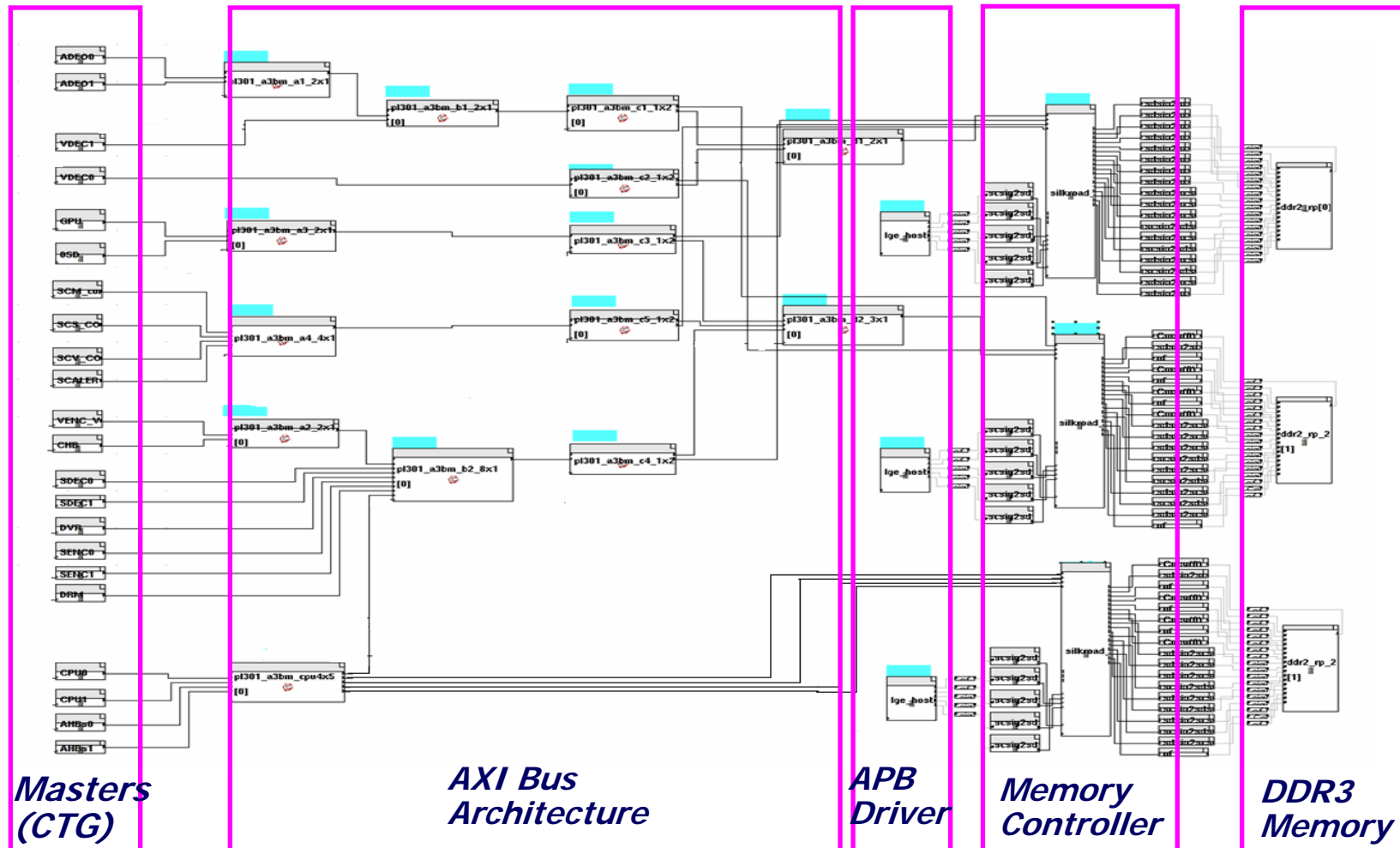


Carbon Component

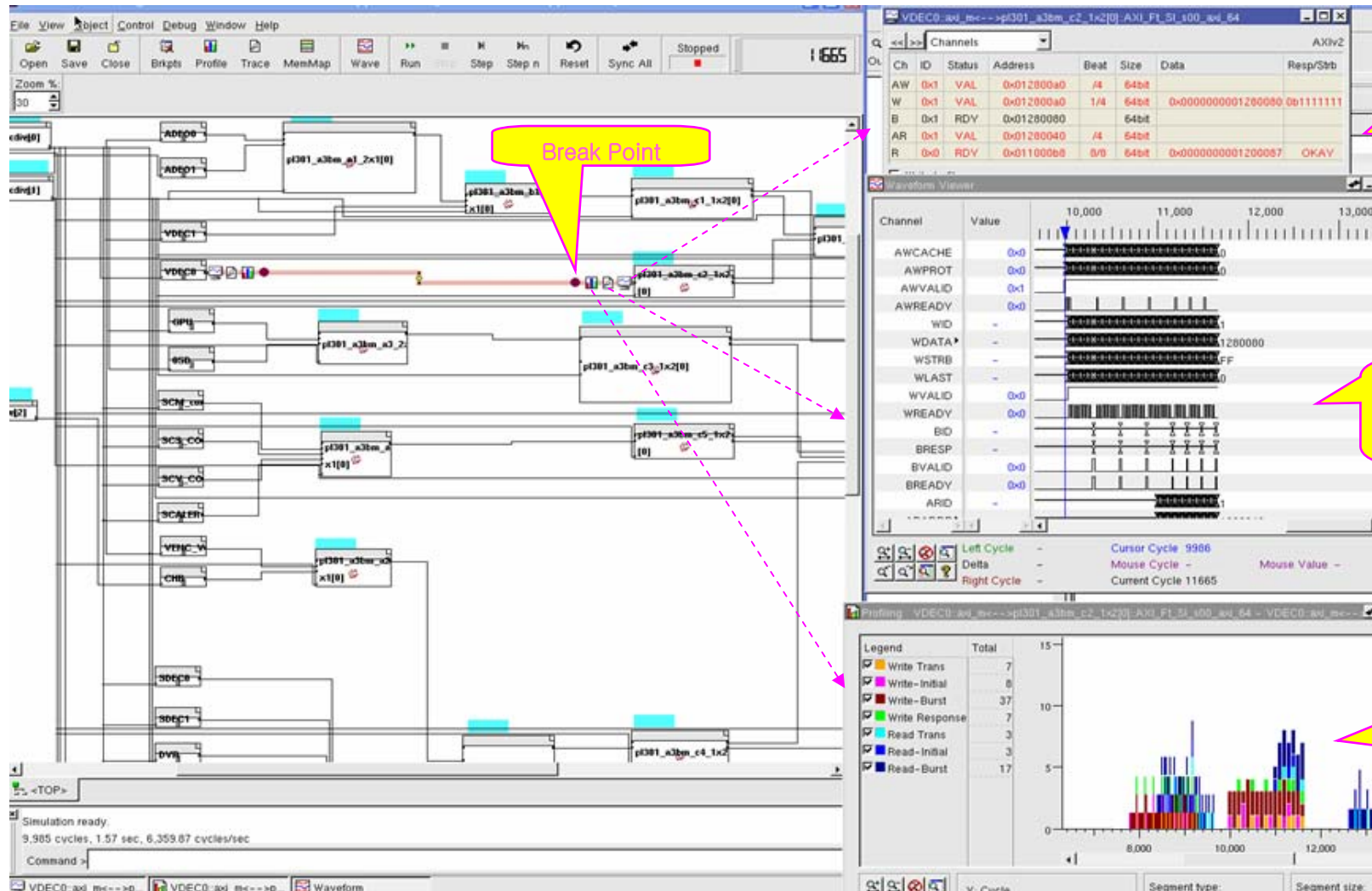
1st phase : LG's own DDR3 memory controller & DDR3 model



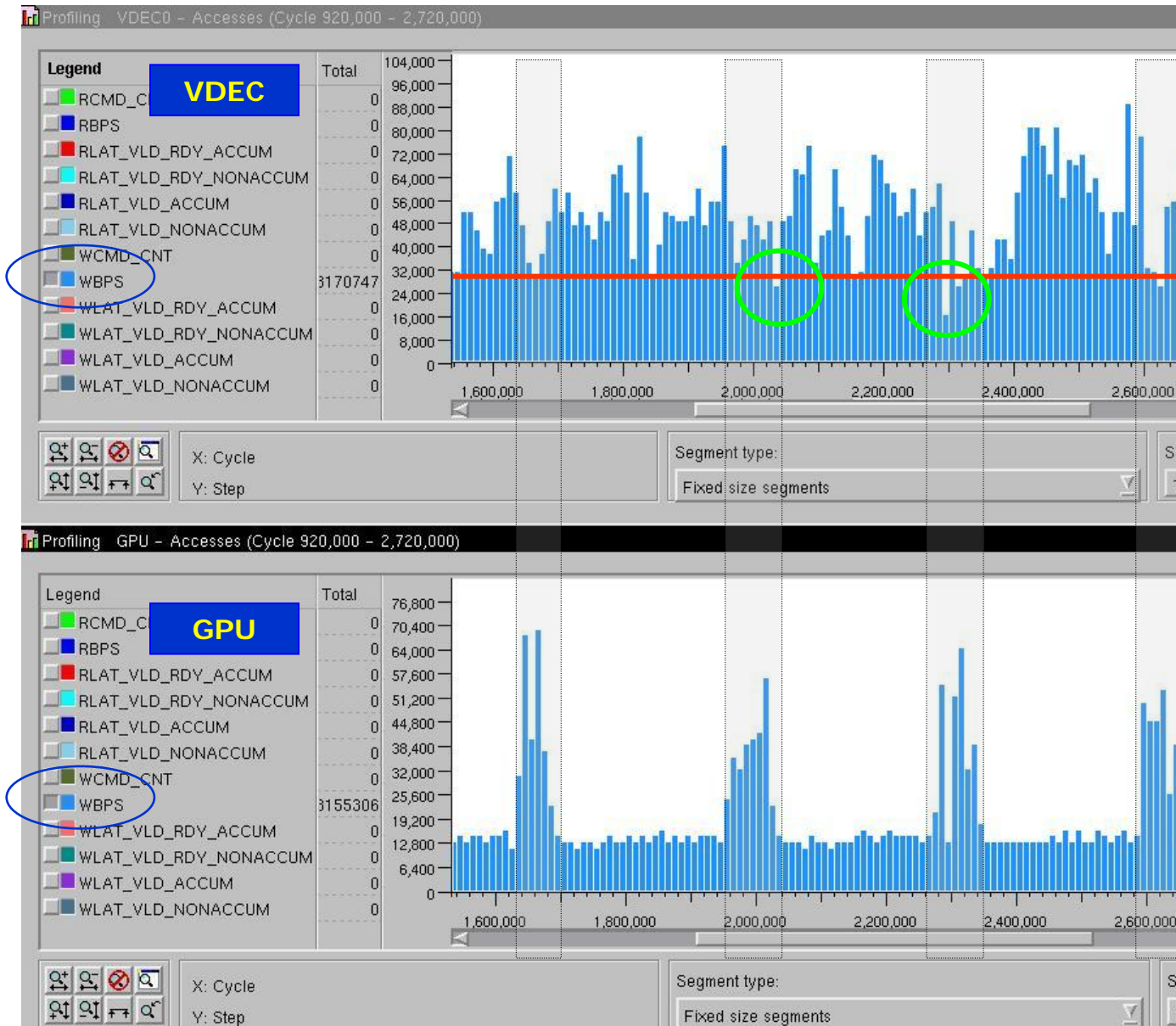
1st phase : Top simulation environment capture.



1st phase : CTG simulation environment [Monitoring, WaveView, Profiling]



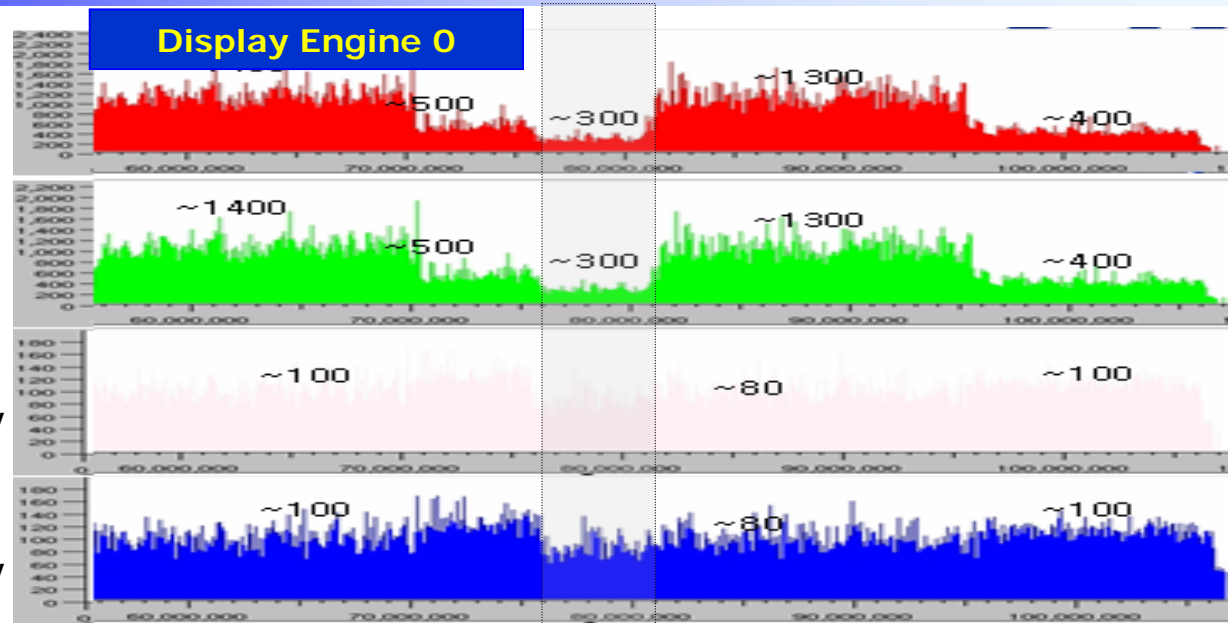
BW Profiling capture 1 : Only one information enabled.



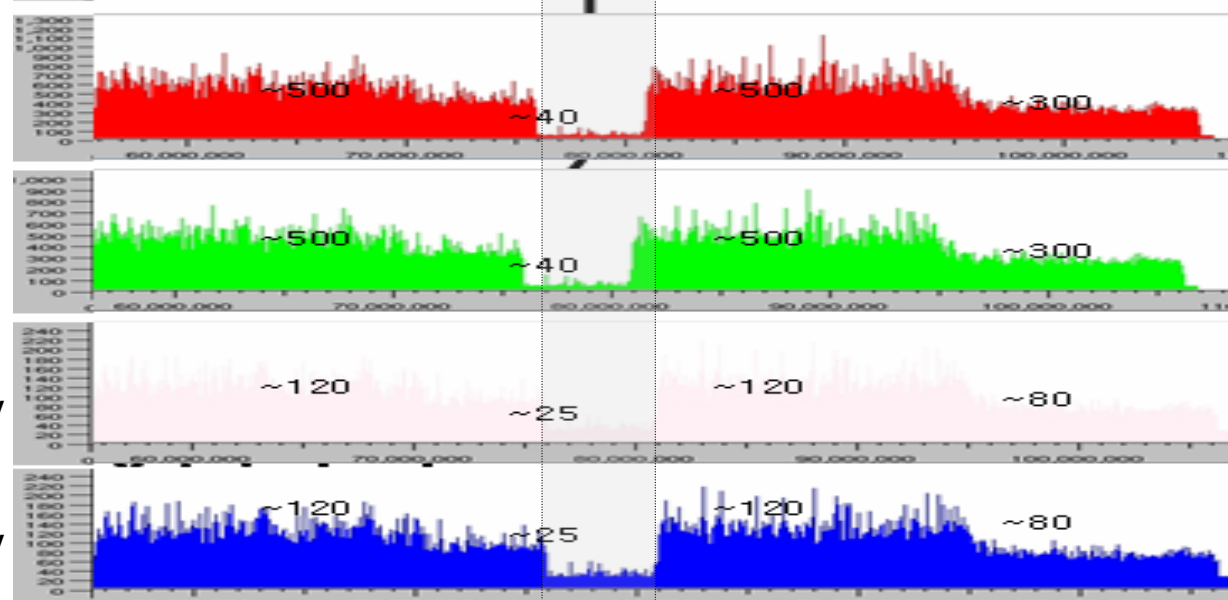
Multi-dimensional Latency Profiling capture.



- Read**
- 1. valid base, including overlapped latency
 - 2. ready base, including overlapped latency
 - 3. valid base, excluding overlapped latency
 - 4. ready base, excluding overlapped latency



- Write**
- 1. valid base, including overlapped latency
 - 2. ready base, including overlapped latency
 - 3. valid base, excluding overlapped latency
 - 4. ready base, excluding overlapped latency



Refer to Appendix about multi-dimensional latency profiling.



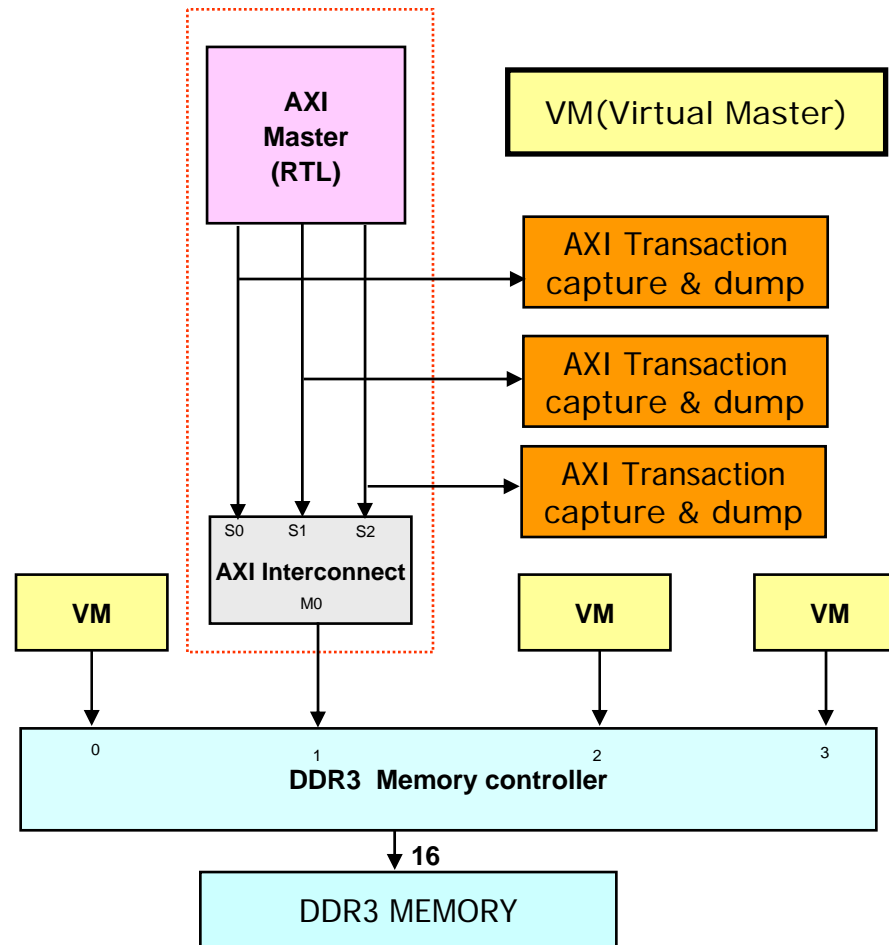
- Master IP optimization
 - **Reduced unfavorable AXI transactions** such as non-burst and random addressing
 - **Increased master's buffer size** to increase **multiple outstanding transactions**
 - **Decreased master's buffer size** which exceeding maximum multiple outstanding limit.
 - Supplemented **reduced operation mode** according to use case scenario.
(ex. changing 10bit -> 8bit processing)

- AXI compatible Interconnect IP optimization
 - Iterative redesigning while considering bandwidth, latency, arbitration, QoS (Quality of Service)., operation frequency, gate count ...etc.
 - To harmonize with the memory controller

- DDR3 memory controller redesign & optimization
 - To support **bandwidth critical & latency critical masters**
 - To find out **optimal arbitration & QoS scheme**
 - To find out **optimal buffer size** & AXI interface port number
 - **To control DDR3 physical memory effectively** according to the master's request

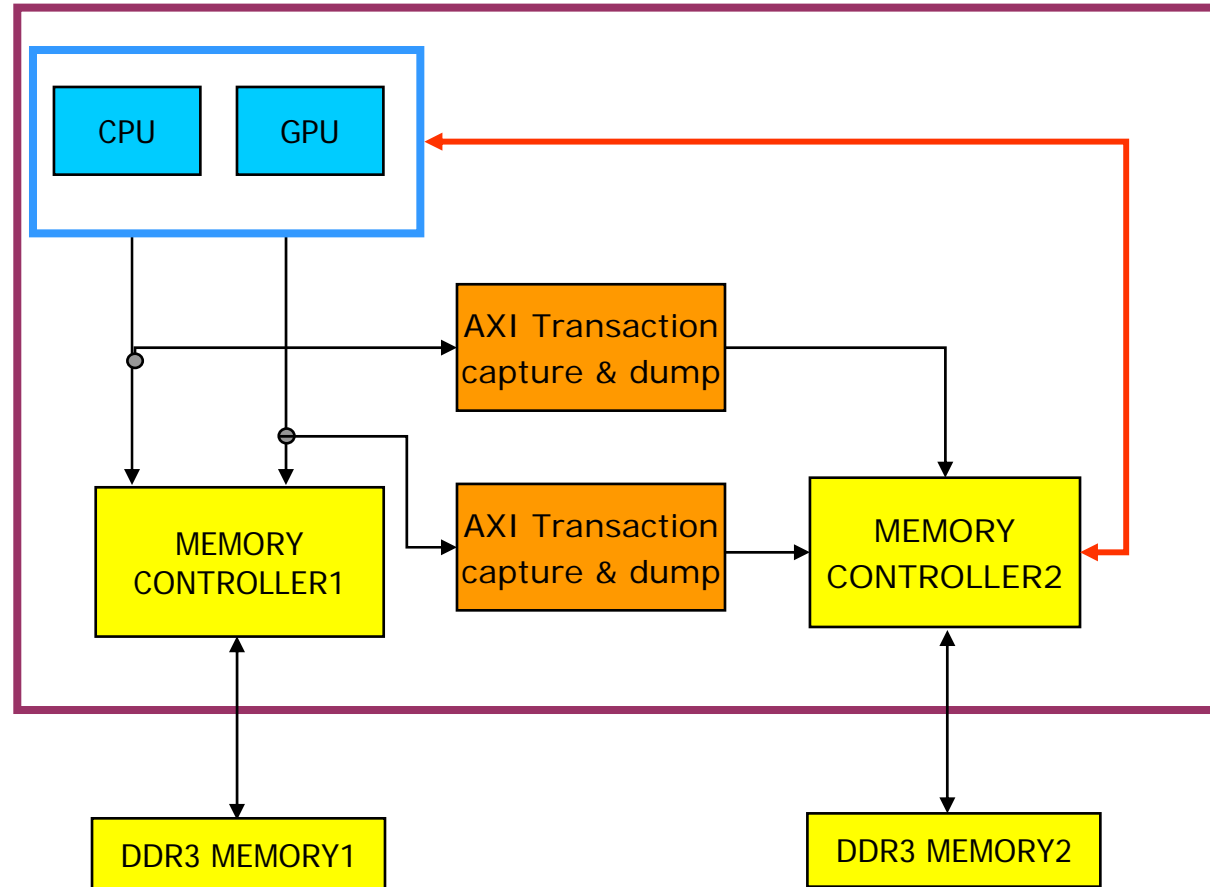


- **Accuracy problems** : Designer provided parameters was **not fully accurate**
 - Masters easy to predict access patterns
 - dependent on fixed video resolution
 - Masters hard to predict access patterns
 - dependent on Operating System & device driver
 - having variable BW according to the inputs stream
 - Outsourcing IPs
- For the 2nd phase, we decided to replay **real AXI transaction traces** which
 - dumped **from RTL simulation**
 - captured **from FPGA board emulation** via AXI transaction capture & dump logic



- Traces were captured from AXI interfaces of IP block and used as AXI traffics for ESL simulation.

2nd phase : FDTG (FPGA Dump file-based Traffic Generator)



- The reason for using 2nd DDR3 memory controller is not to disturb 1st DDR3's memory operation



- This approach presents
 - **architectural insights and visibility**
 - bandwidth profiling, multi-dimensional latency profiling, transaction statistics
 - **faster reconfiguration & simulation time.** (10~100 times faster than RTL simulation time)
 - **minimum resources** in ESL-related work
 - by saving SystemC modeling time for IP blocks.
- This approach can be applied **even without RTL** implementation
 - solution for the “chicken-and-egg problem”
- Thanks to this two-phased approach,
 - memory throughput **improved up to 19%** over initial design in best case.
 - achieved **88% effective DDR3 memory utilization** In best case
 - finished top architecturing **even before all IP's RTL is ready.**
 - **accomplished another chip's top architecture design very quickly** by using this project's result.

References.



- Carbon Model Studio User Documentation by



- Carbon SoC Designer User Guide.



- <http://www.carbondesignsystems.com/>

- AMBA® AXI Protocol v1.0 by **ARM®**

<http://www.arm.com/>

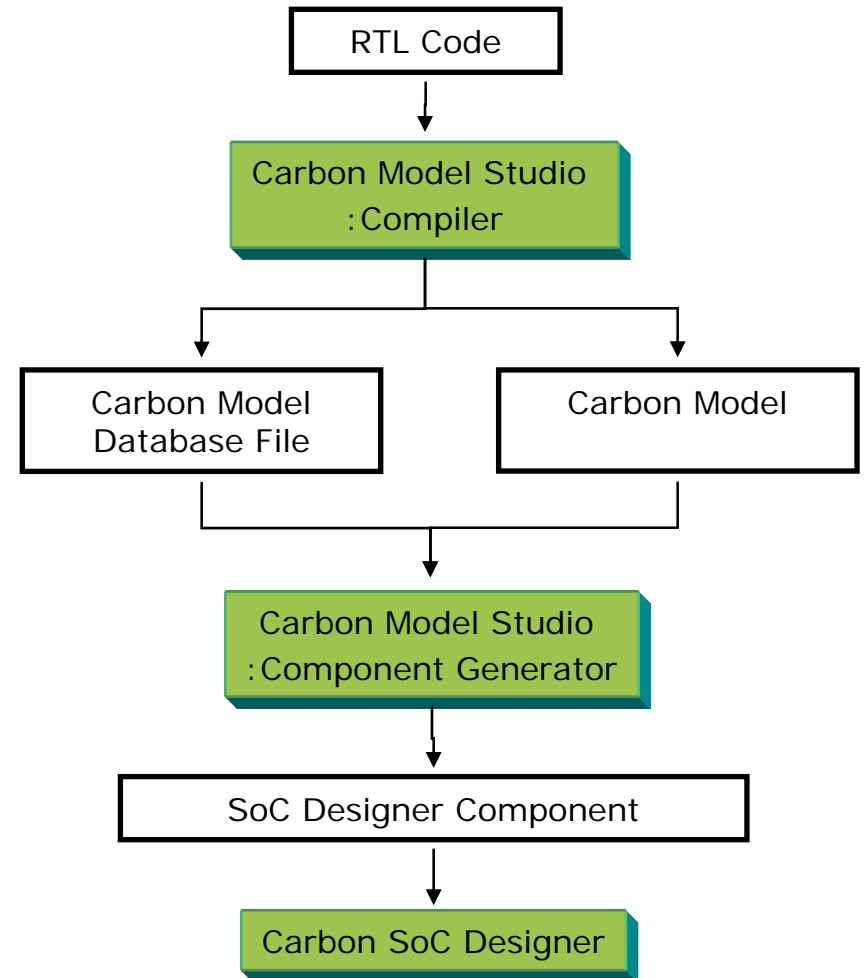


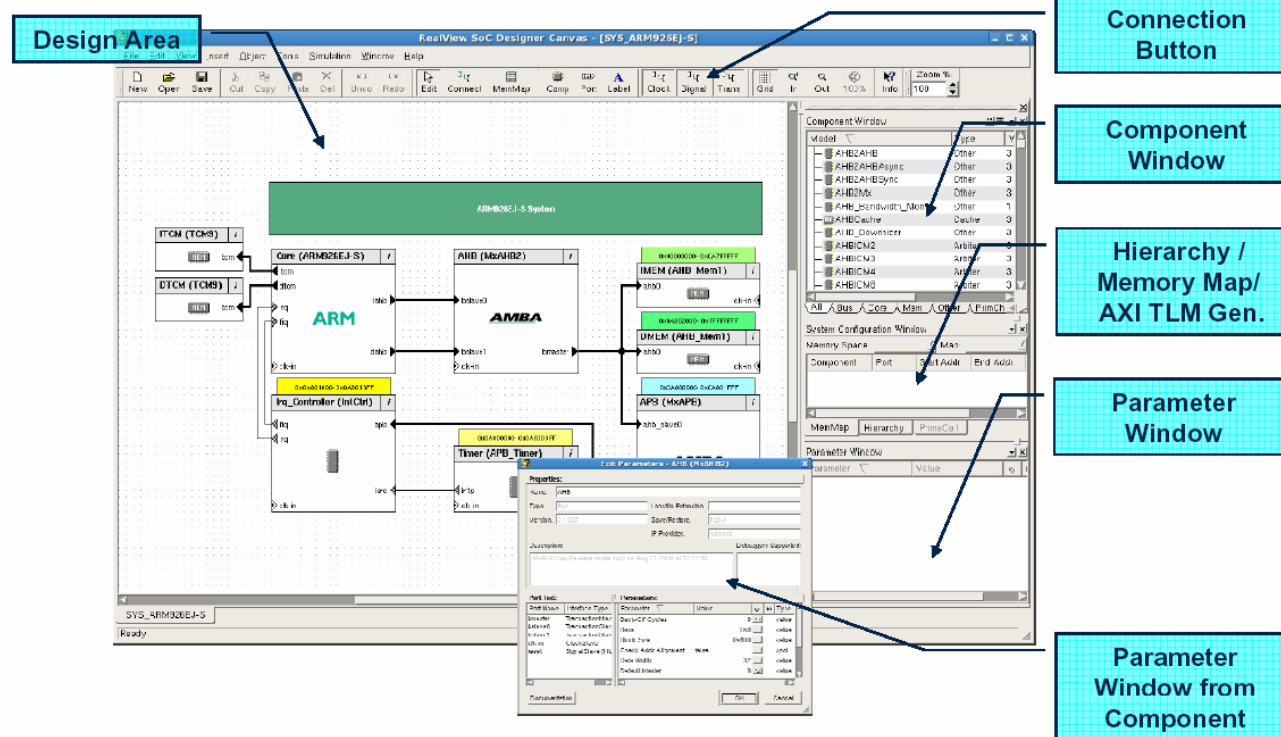
Appendix

- You can use Carbon Model Studio to create a Carbon SoC Designer-compatible component from the Carbon Model, which includes the simulation and debugging interfaces.

Using this process, system engineers use the component in SoC Designer to build a simulatable system and perform the following tasks:

- Architectural exploration
- System and software validation and debug
- Embedded software development





- Carbon SoC Designer is a simulation environment for easy modeling and fast simulation of integrated systems-on-chip with multiple cores, peripherals and memories using C++.
- The SoC Designer systems can be used as standalone simulation models or integrated into HW simulation or HW/SW co-simulation tools.
- Carbon SoC Designer is fully based on the SystemC language. While cycle-based simulation is strongly recommended for speed and ease of debugging, SoC Designer supports running any event-driven SystemC simulations.

1st phase : AXI master modeling : CTG Parameters



CTG_name	CTG component name	
MAX_WID/ MAX_RID	maximum number of AXI ID within CTG. All parameters can be set differently per ID.	
Required BW Parameters	CTG is the AXI Master generating AXI transaction as the speed of bandwidth which was set by user.	
	WBPS / RBPS	required Bandwidth parameter (unit : bps(bit per second))
	OPER_FREQ	operating frequency of CTG (unit : of parameter is Hz)
	AWSIZE/ ARSIZE	maximum number of data bytes to transfer in each beat, or data transfer, within a burst.
Addressing parameters	AXI Master(Video encoder/ Video decoder/ GPU/ CPU) has his special address pattern.	
	W_OUTSTANDING / R_OUTSTANDING	numbers of AXI commands issuing in succession without waiting of previous transaction done.
	RANDOM_WEN / RADDOM_REN	enable parameter when the following address of issuing command is random compared to past address of it.
	AWLEN / ARLEN	Burst length, The burst length gives the exact number of transfers in a burst.
	AWBURST / ARBURST	Burst type parameter this parameter is equal to burst type signal of the AXI protocol
Profiling Parameters	When we simulate Platform in ESL Level, we get Profiling information. The following profiling information a) Measured Bandwidth display b) Measured latency display c) the section which AXI transaction occurs.	
	PROFILE_EN	The profiling enable parameter
Others	... There are more parameters.....	



- **Address pattern parameter examples. ('b' means burst length)**
 - **[wr.incr 4*b8@id0] & [rd.random 8*b4@id1] ('&' means it occurs simultaneously)**
 - random_wen = 0, w_outstanding = 4, awlen = 7, id = 0
 - random_ren = 1, r_outstanding = 8, arlen = 3, id = 1
 - **[wr.incr 4*b8@id0] + [rd.random 8*b4@id0] ('+' means it occurs successively)**
 - random_wen = 0, w_outstanding = 4, awlen = 7, id = 0
 - random_ren = 1, r_outstanding = 8, arlen = 3, id = 0
- **Complex address pattern is possible.**
 - **[wr.incr 5 * (b16 + b4) @ id0] & [rd.incr 1 *(b10 + b3)@id1] & [wr.random 16*b2@id2]**
 - random_wen = 0, w_outstanding = 5, awlen0 = 15, awlen1 = 3, id = 0
 - random_ren = 0, r_outstanding = 1, arlen0 = 9, arlen1 = 2 , id = 1
 - random_wen = 1, w_outstanding = 16, awlen = 1, id = 2
 - **[wr.incr 5 * (b16 + b4) @ id0] + [rd.incr 1 *(b10 + b3)@id0] + [wr.random 16*b2@id0]**
 - random_wen = 0, w_outstanding = 5, awlen0 = 15, awlen1 = 3, id = 0
 - random_ren = 0, r_outstanding = 1, arlen0 = 9, arlen1 = 2 , id = 0
 - random_wen = 1, w_outstanding = 16, awlen = 1, id = 0



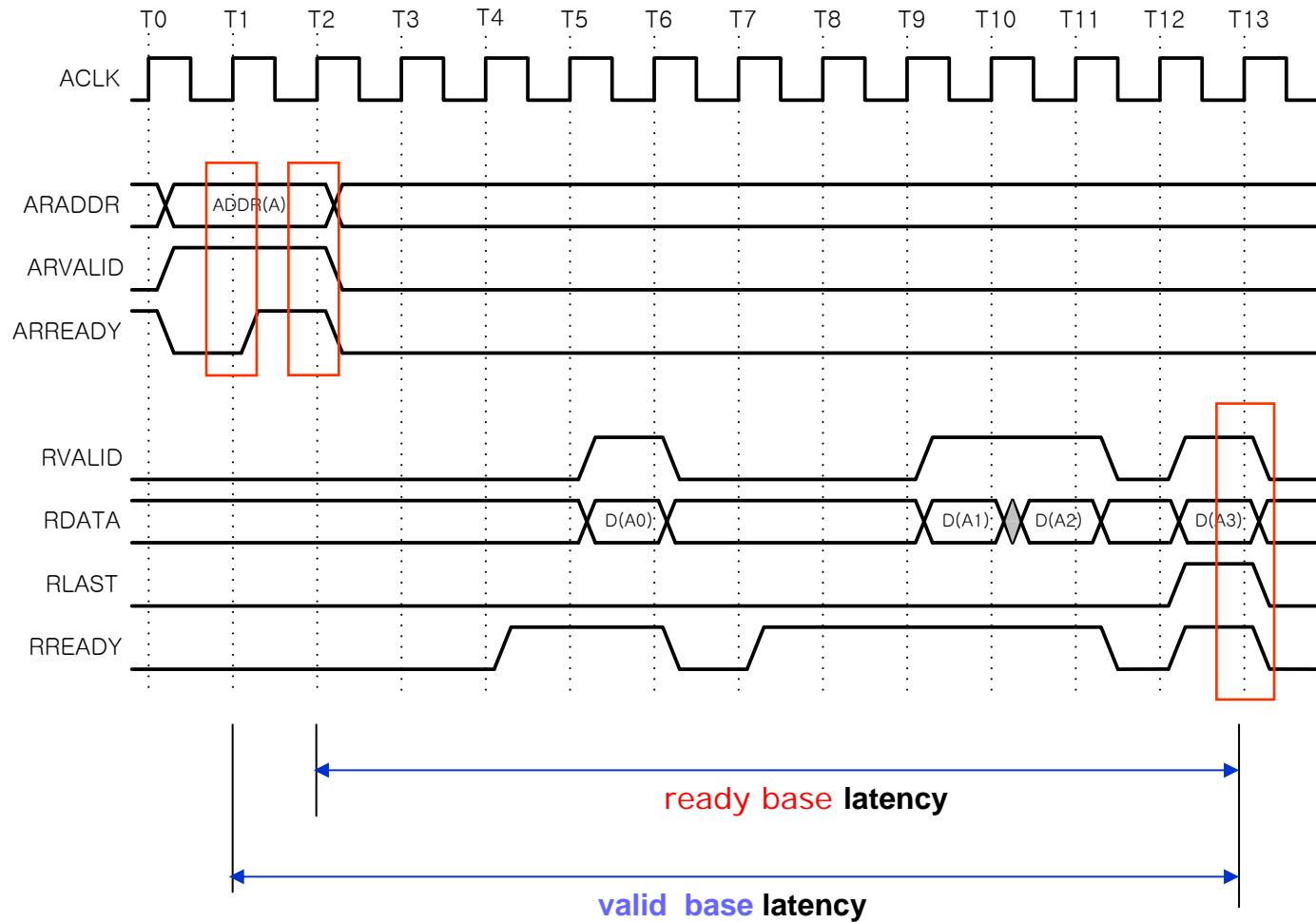
■ LG defined 4 types of latency for latency profiling.

- 1. valid base, including overlapped latency
- 2. ready base, including overlapped latency
- 3. valid base, excluding overlapped latency
- 4. ready base, excluding overlapped latency

Latency : Definition of **valid base** & **ready base** : read example

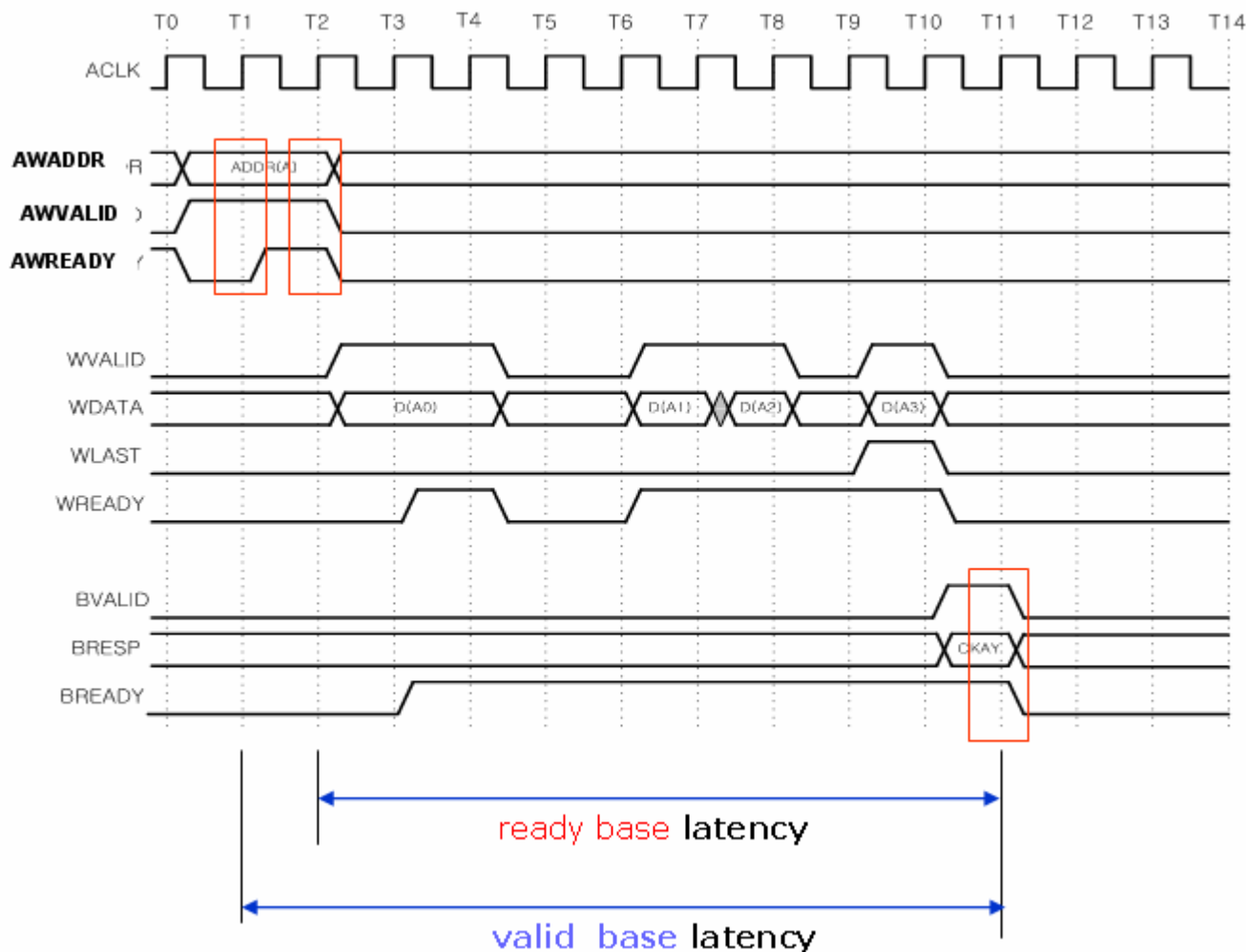


■ Read Burst 4 example



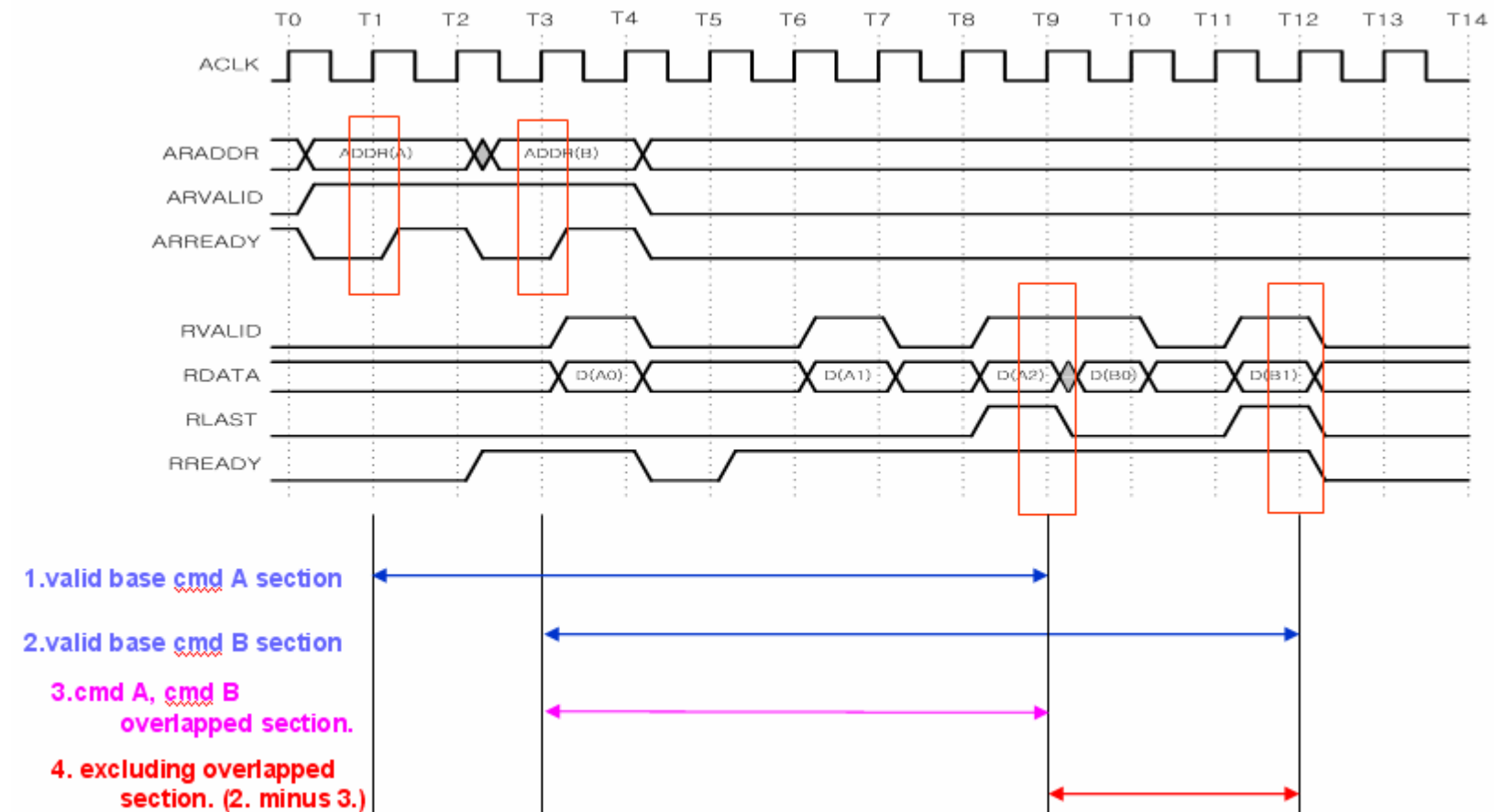


■ Write Burst 4 example





■ 2 overlapping read bursts.



- cmd B's valid base including overlapped latency is '9' (T12-T3)
- cmd B's valid base excluding overlapped latency is '3' (T12-T9)