Data Flow Based Memory IP Creation Infrastructure

Abhilash V Nair, Texas Instruments, Bangalore, India

Praveen Buddireddy, Texas Instruments, Bangalore, India

Tor Jeremiassen, Texas Instruments, Bangalore, India

Rashmi Venkatesh, Texas Instruments, Bangalore, India

Prajakta Bhutada, Texas Instruments, Bangalore, India

***Abstract*— This paper introduces a methodology to enable ease creation of IP and reuse of it across various platforms (Function, Cycle Accurate or Architecture platform). It provides ease way of converting an IP expressed as Data Flow Diagram into SystemC/C++ code.**

I. INTRODUCTION

Performance improvement is one of key goals in defining any next generation SOC. Different ways to enable this are by adding more processing capability in the processing element like DSP or ARM or existing accelerators or adding more acceleration component replacing the software.

Increasing Processing means more bandwidth requirement from the Memory subsystem. It is very important to make sure the application does not become IO bound (IO bound => data not available for processing).

Exploring the Memory subsystem to provide the right bandwidth to the processing element is always very critical. This paper tries to address Memory exploration using Data Flow Based Memory IP Creation Infrastructure.

II. PRESENT METHODOLOGY

A. RTL based Memory Exploration

RTL is enhanced with various configurable knobs which may include cache size, buffer size etc. This method works fine if the amounts of changes are limited to a few subsets and does not involve major change in the memory architecture.

Some of limitations with this approach are listed below

- Memory system evaluation using real usecase would be slow. Running a full system usecase on RTL may take days to complete
- Major changes in the memory architecture cannot be done
- Amount of effort to modify and maintain the correctness of the model is high.

.

B. Use existing Simulation framework like GEM5/Ruby

GEM5/Ruby provides a good framework to implement a detailed simulation model for the memory subsystem.

Some of limitations with this approach are listed below

- Enhancing the code for any memory exploration change needs a good knowledge of the infrastructure. Learning curve of infrastructure is steep.
- Not intended to create cycle or functional accurate models.
- Productization of simulator for software development involves huge effort.

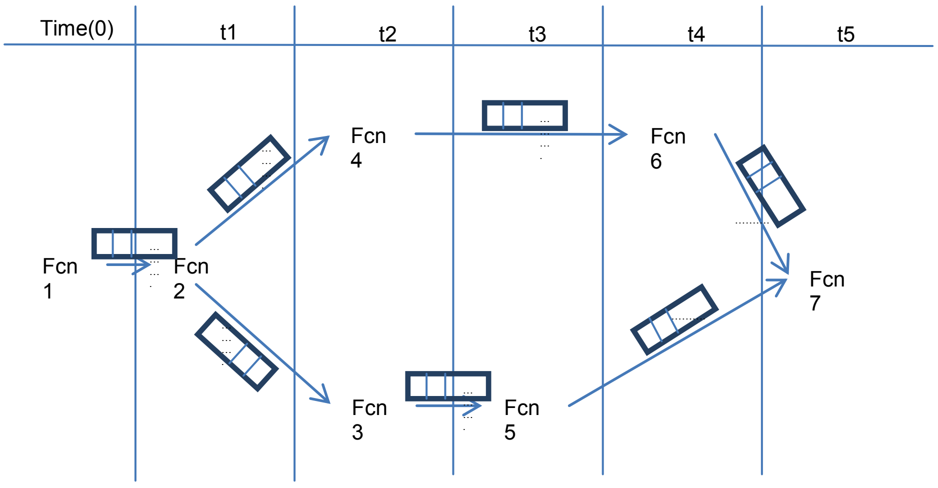
C. *Paper based Calculation*

This has been the traditional method of doing exploration when there is no architecture exploration tool. It involves obtaining data from the existing SOC and interpreting the data based on proposed memory enhancement. The data correctness is always under question due to non-availability of simulation result.

III. Memory Architecture exploration using Data Flow Based Memory IP Creation Infrastructure

The Infrastructure aimed at easily creating a memory hierarchy model. It provides

- Mechanisms to refine the implementation to match a particular architecture or explore architectural tradeoffs.
- Gives the Architect or Designer of a memory subsystem a standard methodology to implement the model in a structured way.
- Migration path where a single model can be evolved to satisfy different requirements (Cycle approximate, Functional accurate, Architecture) with a high degree of reuse and improved R&D efficiency



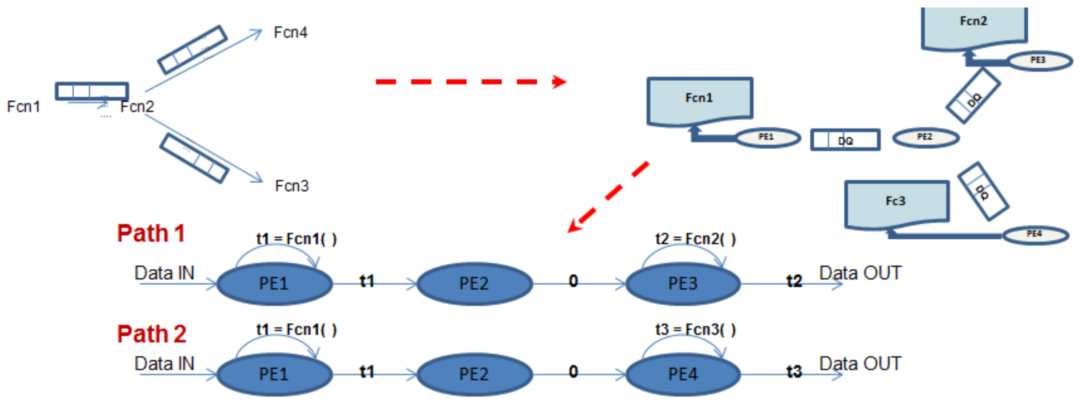
The above Figure shows Data Flow Diagram (DFD) for a simple IP. IP's (including Memory IP) are expressed as DFD using the Infrastructure. This helps to capture the different basic functional elements and the path use to connect them. The Nodes are the functional element and Edges define the flow of data.

The Nodes captures the functionality as well as provides the time duration to complete processing. These nodes are termed as Processing Element (PE).

The Edges represents flow of data between the PEs. It captures the maximum outstanding requests in the path and holds the outstanding request between the PEs. It simulates the time duration, return by the IN PE, before calling the OUT PE. These edges are termed as Delay Queue (DQ)

* Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors).*

IV. DETAIL EXAMPLE USING THE INFRASTRUCTURE



The above diagram tries to show how a sample DFD (with timing and data) is represented in the Infrastructure. The IP has three Processing Elements PE1, PE2 and PE3. Data flows from PE1 to PE2 or PE3 based on the outcome of the processing. Hence the data can take two path based on the processing result.

```
 /* Creating the PE and DQ */
PE1= new PE<data_type, ..........................> ("PE1", num_of_input, num_of_output);
.................................................................................
/* Attaching Function */
PE1->add_semantic_function(boost::bind(&class_x1::fcn1,...));
PE3->add_semantic_function(boost::bind(&class_x1::fcn3,............));
....................................................................................
 /* Binding the PE's */
PE1->bind(PE2,PE1_TO_PE2,max_capacity);
PE2->bind(PE2,PE2_TO_PE3,max_capacity);
PE2->bind(PE2,PE2_TO_PE4,max_capacity);

/* Container for PE functionalities */
Class class_x1 {
              ................
              direction_t fcn1(data_in, latency1, ........................)
              direction_t fcn2(data_in, latency2, ........................)
}
```

The above figure shows code corresponding to the DFD. The code involves

- Creating of PE and DQ and attaching the Processing Function
- Connecting the PEs based on the Data flow. Also specifying the maximum outstanding transaction between the PEs

V. OTHER FEATURES SUPPORTED BY THE INFRASTRUCTURE

A. Memory IP Building Blocks

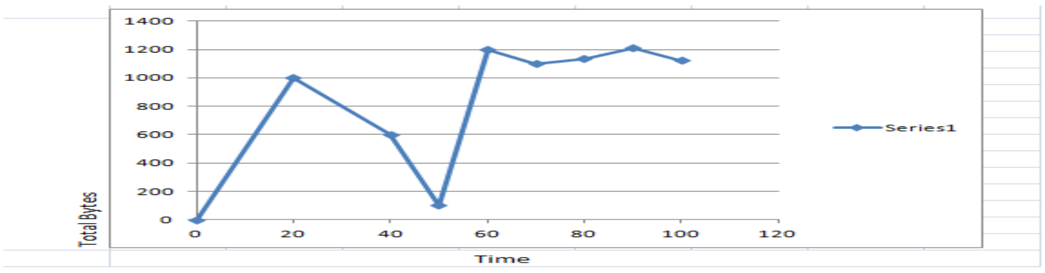
The infrastructure support below utilities/building blocks

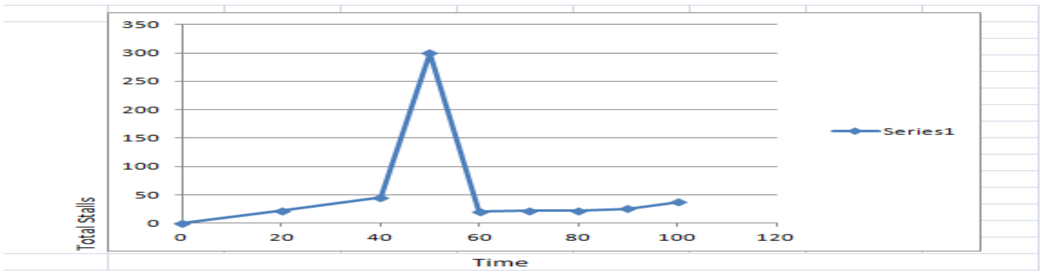
- Generic cache objects with configurable size, line size, way, replacement policy and hit/miss latency.
- Generic memory object with configurable size and latency
- Adapters to interface with different protocol like CBA, AXI, OCP
- Write Buffer with configurable size and write merge algorithm
- Prefetch Engine with Configurable buffer size and address generation algorithm

The building blocks are readily available for the user of the Infrastructure. The user can focus on the development of custom processing element in the IP

B. Statistics

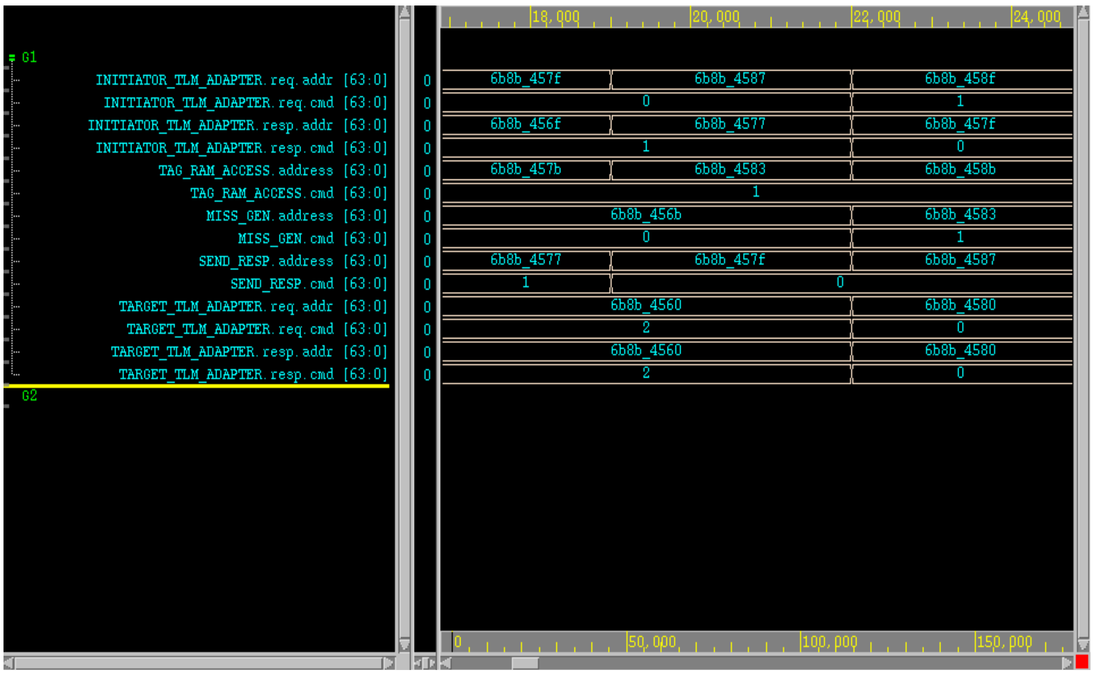
Each PE contain two major events viz total number of transaction processed and total stall cycles due to output DQ reached max capacity or others. Both these event can be traced over time to give insight into the PE's performance.





The above diagram shows example graphs plotting the for total bytes and total stalls over time.

VCD file with information of flow of data/transaction over the PE and DQ w.r.t to time. Below is a snapshot of a VCD trace. It provides a very strong tool to debug further the bottleneck in specific path



```
                                                 18,000        20,000        22,000        24,000
G1
INITIATOR_TLM_ADAPTER.req.addr [63:0]    0   6b8b_457f    6b8b_4587                  6b8b_458f
INITIATOR_TLM_ADAPTER.req.cmd [63:0]     0                0                          1
INITIATOR_TLM_ADAPTER.resp.addr [63:0]   0   6b8b_456f    6b8b_4577                  6b8b_457f
INITIATOR_TLM_ADAPTER.resp.cmd [63:0]    0                1                          0
TAG_RAM_ACCESS.address [63:0]            0   6b8b_457b    6b8b_4583                  6b8b_458b
TAG_RAM_ACCESS.cmd [63:0]                0                          1
MISS_GEN.address [63:0]                  0                6b8b_456b                  6b8b_4583
MISS_GEN.cmd [63:0]                      0                0                          1
SEND_RESP.address [63:0]                 0   6b8b_4577    6b8b_457f                  6b8b_4587
SEND_RESP.cmd [63:0]                     0   1                          0
TARGET_TLM_ADAPTER.req.addr [63:0]       0                6b8b_4560                  6b8b_4580
TARGET_TLM_ADAPTER.req.cmd [63:0]        0                2                          0
TARGET_TLM_ADAPTER.resp.addr [63:0]      0                6b8b_4560                  6b8b_4580
TARGET_TLM_ADAPTER.resp.cmd [63:0]       0                2                          0
G2
                                             0        50,000        100,000        150,000
```

VI. CONCLUSION AND FUTURE WORK

The proposed methodology accelerates various pre silicon activities from architecture exploration to software development of Memory IPs. It enables Architect or designer to create the model with minimal knowledge of SystemC simulation. It provides quantitative data early in the cycle for informed decisions.

It enables re-use of model across different stages in development starting from Architecture definition to software development/benchmarking

Other key capability which can be extended to enable critical IPs exploration

- Feasibility of creating Arch model for Interconnects
- Support of exploration around cache coherency to select the right coherency method (MSI, MESI, ….)
- Extending to create Non-Memory specific IP models.