Data Flow Based Memory IP Creation Infrastructure

Authors: Abhilash V Nair, B. Praveen Kumar, Tor Jeremiassen, Prajakta Bhutada and Rashmi Venkatesh
Outline

• Problem Statement
• Present methodology & Limitations
• Proposed methodology
• Novelty
• Case study: Prefetch Analysis
• Case study: DSP Memory and Low Level Cache Arch
• Conclusion
• Questions
Problem Statement / Introduction

• Performance improvement is one of key goals in defining any next generation SoC. Different ways to enable this are
  – Adding more processing capability in the processing element like DSP or ARM or existing accelerators.
  – 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.
Present Methodology : 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 amount of changes are limited to a few subset and does not involve major change in the memory architecture.

• Limitations
  – 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 can not be done
  – Amount of effort to modify and maintain the correctness of the model is high.
Present Methodology: Simulation framework like GEM5/Ruby

- GEM5/Ruby provides a good framework to implement a detailed simulation model for the memory subsystem.
- The following picture* shows a high-level view of the main components in Ruby. For more reference http://www.m5sim.org/Ruby

Limitations
- 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.

* from the GEMS tutorial in ISCA 2005
Present Methodology : Paper based Calculation

• This has been the traditional method of doing exploration when there is no architecture exploration tool

• Involves obtaining data from the existing SOC and interpreting the data based on proposed memory enhancement.

• Limitation
  – The data correctness is always under question due to non availability of simulation result.
Proposed methodology: Memory Architecture exploration using Data Flow Based Memory IP Creation Infrastructure

- **Infrastructure**
  - aimed at easily creating a memory hierarchy model
  - provides mechanisms to refine the implementation to match a particular architecture or explore architectural tradeoffs

- It gives the Architect or Designer of a memory subsystem a standard methodology to implement the model in a structured way.

- It provides a 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 show Data Flow Diagram (DFD) for a simple IP. The nodes are the functional element and edges defines the flow of data. 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 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.
- captures the maximum outstanding requests in the path.
- holds the outstanding request between the PEs.
- simulates the time duration, return by the IN PE, before calling the OUT PE. These edges are termed as Delay Queue (DQ).
Proposed methodology: Example of IP modeled using the Infrastructure

- The below diagram tries to show how a sample DFD (with timing and data) is represented in the Infrastructure.

Each PE has the functionality associated as shown in the DFD.
- \( t_x(t_1, t_2, t_3) \) represent timing. For example, \( t_1 \) represents the time taken by PE1 to process the data before pushing into PE2.
Proposed methodology: Example of IP modeled using the Infrastructure

- Pseudo code corresponding to the previous sample DFD is shown below

```c++
/* 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, ……………………..)
}
```

- Data Type: The transaction type which is processed by the PE
- Num_of_input: Number of input to the PE.
- Num_of_output: Number of output from the PE.
- class_x1::fcn: Processing function attached to PE
- max_capacity: Maximum number of outstanding transaction in the DQ
- latency: Time taken by the PE to process the data_in. This would be specified by the Processing function.
Proposed methodology: Example DFD for Cache modeled using the Infrastructure
Proposed methodology: Utilities Details

Utilities contain the following building blocks specific to IP like Memory.
- Generic cache object 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
  - TLM2.0 Base protocol adapters
  - CBA protocol adapter
- 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.
Proposed methodology: Statistic supported by the Infrastructure

• Each PE contains two major events
  • Total number of transactions processed
  • Total stall cycles due to output DQ reached maximum capacity or others
• Both these events can be traced over time to give insight into the PE’s performance
Proposed methodology: Statistic supported by the Infrastructure

• VCD file with information of flow of data/transaction over the PE_DQ w.r.t to time. Below is a snapshot of a VCD trace

• This provides a very strong tool to debug further the bottleneck in specific path
Novelty of the Proposed Solution

• Ease of Use
  – It provides an easy and standard model developing methodology. The user of this platform need no knowledge of the Infrastructure. The user is expect to write simple “C” like functions capturing the processing and timing details.

• Faster Arch model creation/exploration
  – Model development effort is much less because of the Infrastructure and existing building blocks
  – It enables to play with configurable parameters with minimum effort during exploration phase

• Better Performance Analysis
  – The statistics like VCD trace, bandwidth graph and event counters like cache hit/miss enable performance analysis

• Model Re-use
  – The same Architecture model can be enhanced to create cycle approximate or functional IP model. 100% code reuse.
    • Separating Functionality (PE) and Timing aspect(DQ) of the IP is key enabler for reuse
Case Study: Prefetch Analysis for TI Microcontroller

• Problem Statement: Analyze the Prefetch buffer requirement for the Instruction path for TI Microcontroller. Study the impact using Coremark benchmark

• Solution:
  • Created a Prefetch IP using the Framework
    • Provide configurability for Prefetching Algorithm, Line size of prefetch buffers and Number of Prefetch buffer entries
  • Obtained trace for the Program access from the RTL and replayed over the Prefetch IP
  • At the target side Latency memory was connected to mimic Flash memory latency

• Result:
  • Provide key feedback to the Architect team on the Prefetch algorithm and size requirement
  • Model development and Analysis could be done in a quick timeframe. Closed on the requirement down to result within a span on 1 month
Case Study: DSP Memory and Low Level Cache Arch model for System Level usecase analysis

- Problem Statement: Study the impact of new architecture change done for DSP Memory Subsystem and Low level cache for system level usecases

- Solution:
  - Created architecture model using the Framework
  - Identified and provide major configurable Knobs to provide feedback to the IP architect

- Result:
  - Input for Arch model was the DFD created by the Architect. The SystemC/C++ model had one to one correspondence with each block in the DFD
  - Achieved close to 100% accuracy as was expectation from the Architecture specification. During the process help correcting the specification numbers based on simulation
  - Development and debug time reduced compare to previous effort. Fully accurate L1, L2 and LLC cache model developed within a span of 3-4 months compare to more than an year which was taken earlier
Conclusion

• Conclusions
  – 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
  – 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

• Limitations and future work
  – Study the 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.
QUESTIONS