Assigned: November 1st 2012
Due: Tuesday, 12/18/2012 7:00 pm EST
Final Firm Deadline after Optional Automatic Extension: December 20, 2012, 5 pm EST
Final Grade Weight: 60%

The objective of this project assignment is to use all the technologies you have learned in this class and throughout your undergraduate degree (for ABET certification requirement.) The project will be worth 60% of your grade and is the centerpiece of this class. You will understand high-level requirements; write design specifications, write random test benches and write a final report summarizing your design efforts. The project will be carried out in groups of four or five, and you have a choice of four projects to choose from:

(a) A 4x4 on-chip interconnection network
(b) A 2-way out-of-order issue superscalar MIPS pipeline with an issue queue of size 16
(c) A Bitcoin miner
(d) A memory controller

If you want to do a different project from the one listed here please email me your proposal and we will set up a time to discuss this further.

All projects will have three intermediate checkpoints that will lead you towards the completion of the project and it is absolutely essential that you meet the goals for each of the checkpoints. The checkpoints are:

1. Project proposals, due Thursday **Nov 8** (5% of class grade)
2. Specification: due Thursday **Nov 15** (15% of class grade)
3. Detailed Microarch/Verification Review due Tuesday **Dec. 4** (20% of class grade)
4. Implementation and final report due Monday **Dec 20** (20% of class grade)

The work tasks for each phase is listed below:

Phase 1: *Project proposal (5%): November 8, 2012*

The goal for this checkpoint is to ensure that you have a team in place, assign responsibilities for individual team members, and understand the design that your team is going to build.

The deliverable for this deadline is a design document. You will meet with me on Nov 10 (Doodle poll will be set up soon) for a review.

- Create the following *outline* for your unit design document and *populate the first two sections.*
    - Your design document should have the following 10 sections:
        - Section 1: Title page

        - Project title
        - Team name
        - Project members
        - Project member duties
            - Elect a Design Master
            - Elect a Verification Master
    - Section 2: Design Overview (A simple plain English description)
    - Section 3: Unit Level interfaces
    - Section 4: Subunit partitioning and interfaces, Test harness structure
    - Section 5: Microarchitecture design
    - Section 6: Verification strategy
    - Section 7: Performance estimates
    - Section 8: Area estimates
    - Section 9: Bugs, Coverage,
    - Section 10: Document revision history

Phase 2: *Specification Review* (20%) November 15, 2012

Four work items need to be completed for this phase. First, all team members will collaborate to identify the units and sub-units in the design. Second, the design team will build a top-level SystemVerilog file describing the unit-level interfaces. Third, the verification team will build an empty test harness that compiles with a top-level file, and simulates for 10000 clock cycles. Finally, all team members will collaborate to complete Sections 3 & 4. You will meet with Simha for a specification review (Doodle poll will be set up).

Phase 3: *Detailed microarchitecture and verification review* (15%) December 4, 2012

For this phase, the design team will work out details of the microarchitecture for their design units including pipelining and stall strategies. The verification team will work on verification test bench parameters and identify all design corner cases. Both sub-teams will collaborate on an incremental design and verification strategy. The deliverables for this phase includes the updated design document (Sections 5,6,7,8), an Excel sheet (template will be handed to you *after* the specification review) highlighting the steps involved in the incremental bring up strategy and slides for the review describing the microarchitecture and verification strategy.

Phase 4: *Implementation and debugging* (20%) December 20, 2012

Following the incremental strategy your team developed in the previous phase, design and verify your design. Then synthesize your design using dc_shell. The deliverables, in addition to the synthesized design, include coverage metrics, timing information, area reports, and power reports. Also include a report describing the bugs encountered and fixes. This will go in section 9 of your design document. You will receive final comments over email.

All documentation and source code is due on December 20, 2012 at 5 PM EST.

Project I: On-chip interconnection Network

BACKGROUND AND BASIC CONCEPTS:

Historically, broadcast based busses have been used for communication within a chip. With the increasing number of processing elements on a single chip and increasing wire delays, bus-based systems do not scale very. The past few years has seen the advent of networks-on-chip (NOC) as replacements for bus-based communication. NOCs provide efficient communication and reduced design complexity when dealing with increasing scale.

The following are some commonly used NOC terms:

Packet: A packet is a unit of message passing. A message comprises of one or more packets. The flow control for a packet-switched network may be implemented at the packet-level or at the flit-level.

*Flit: (**Fl**ow control dig**it**):* A packet may be composed of one or more flits. A flit is the basic data unit used for flit-based flow control. That is, the flow control mechanism operates on flits and not packets.

Wormhole routing: It is a type of flit switching used in a flit-based flow control mechanism. A flit can be transmitted to the next router if there is space available to accept it. The flit need not wait for its associated flits (that make up the complete packet) in order to be sent out.

The router examines the header packet and decides the destination of the flit. All body flits that follow this header flit are simply routed the same way the header flit was routed. Hence, the name "wormhole".

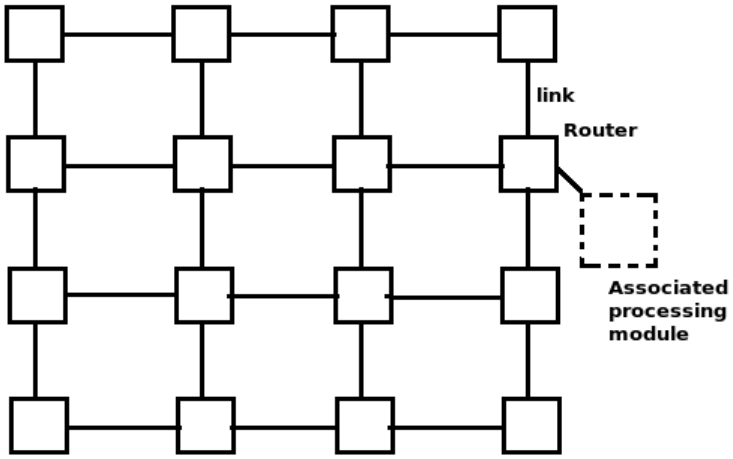
Dimension-ordered routing: While routing data to nodes in a two-dimensional grid, we assign X and Y co-ordinates to each node. Dimension-ordered routing specifies a preferred direction of routing when a choice has to be made. For example, we describe a scenario and how it is handled by Y-X dimension-ordered routing,

When a message is to be sent from (X1, Y1) to (X2, Y2), the message is first sent along the Y-axis to some node (X3, Y2). It is then transmitted along the X-axis to node (X2, Y2).

Credit-based flow control: A credit-based flow control mechanism is one that ensures that the transmitting router sends data only when there is a free slot available in the destination router to accept it. This is implemented by a form of credit communication in which the transmitting router maintains a count of the number of free slots ("credits") in the input buffer of each destination (a separate count for each).

A mechanism of credit communication is required. Each input buffer transmits a count of the number of slots consumed whenever data is consumed (either sent out or consumed locally). This is monitored by source routers that transmit data to this router. Each router also receives credit values that correspond to number of buffer slots consumed by their destination routers. When a piece of data is sent out, the router decrements the credit count of the destination router. When it receives a credit count indicating that some slots have been freed, this counter is incremented again.

YOUR PROJECT:



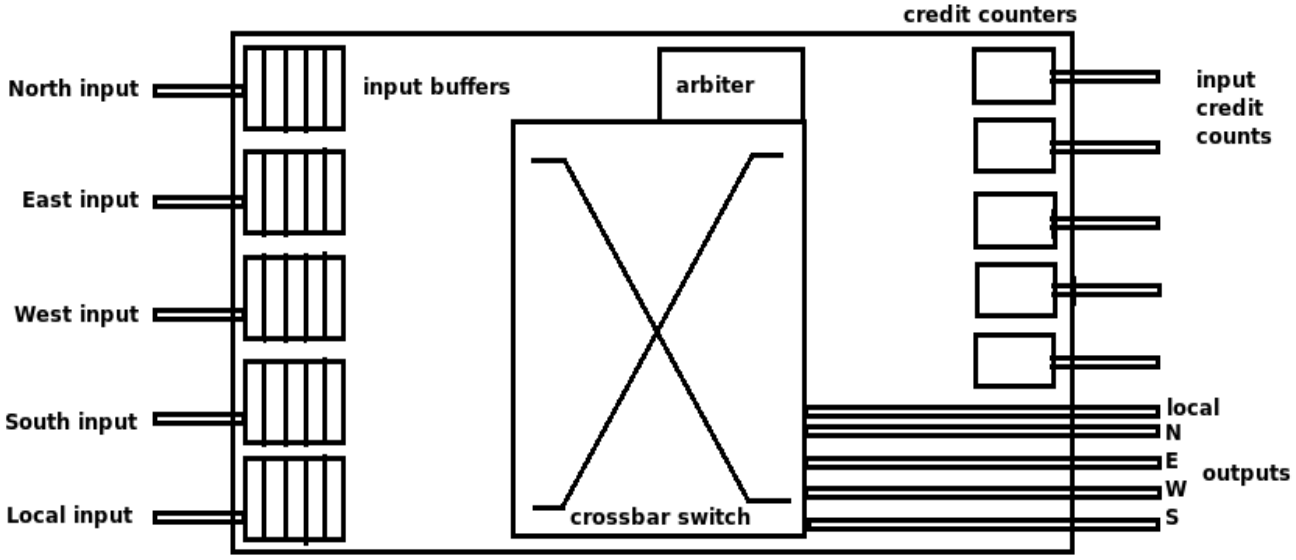
In this project, you will build a 4x4 mesh network using completely synthesizable SystemVerilog HDL. You will also verify your design using SystemVerilog based testbench. The design specifications are as follows:

1. All the routers implement packet-switching with flit-based flow control.
2. The routing is Y-X dimension-ordered and credit-based.
3. In the basic part of this project, you will design routers that process single-flits that are only composed of a header. That is, the header flits do not have any body flits associated with them. In the "challenge" portion of the project, you will implement a Wormhole routing mechanism on top of the basic routing mechanism that can handle 4 body flits which follow each header packet. Note that routing is performed only on header packets – the body packets simply follow through the "hole" created by the header packet.

The header format is as below:

Bits 15 to 8	Bits 7 to 4	Bits 3 to 0
EMPTY	X co-ordinate	Y co-ordinate

High-level description of the router:



Each router has 5 input ports from which it receives data – North, East, West, South and local. The local port is for receiving data from the local processing module associated with the router. Each of these ports has a 5-flit deep input buffer. In the basic case, this means that they can hold five independent header flits. In the challenge part, the input buffers hold a header flit and the 4 trailing body flits.

Internally, the router consists of a crossbar switch that routes data from a particular incoming port to an output port in a *different* direction. This prevents data from being routed from a port in the same direction to itself – which would create a deadlock (for example, data arriving into the North input port cannot be routed to the North output port). Additionally, if two input flits from two different directions need to be routed in the same direction, arbitration is provided by an arbiter module.

Each router also receives 5 input signals, one each from its neighbors (and the local module). These are used to indicate the number of credits available in the respective input buffers of those routers. The router also has an associated counter for the input buffer of each neighbor (and the local module). This indicates the number of free credits available for use in the buffer. It is initialized to the depth of the input queue at the receiver. When a flit is sent out, the counter is decremented. When it receives a signal from the receiver that a flit has been consumed, the credit counter is incremented.

Similarly, each input buffer of the router sends out a value that indicates the number of values consumed in the input buffer (and the resulting number of freed credits).

There are no buffers on the outputs of the routers. The flits are simply routed to the output wires for the input buffer at the receiver to accept.
The above section described a typical router that has neighbors in all four directions. The routers at the edges have one neighbor less in one direction. It would be possible to create an inefficient design in which all 16 routers are designed the same with the edge routers having one port unconnected. However, we want you to optimize for area by modifying the edge routers to actually have one less port.

Project II: Superscalar MIPS pipeline

The objective of this assignment is to build a superscalar MIPS pipeline.

BACKGROUND:

Basic techniques to exploit instruction level parallelism such as loop unrolling, branch prediction, dynamic scheduling, and speculation only suffice to reduce the Clocks Per Instruction (CPI) to a minimum of 1. If we seek to improve performance further, we must reduce the CPI to less than 1. Multiple-issue processors help us achieve this goal by permitting multiple instructions to issue in a clock cycle.

Different types of multiple-issue processors are summarized in the table below.

Common Name	Issue structure	Hazard detection	Scheduling	Distinguishing Characteristic
Superscalar (static)	Dynamic	Hardware	Static	In-order execution
Superscalar (dynamic)	*Dynamic*	*Hardware*	*Dynamic*	*Some out-of-order execution, but no speculation*
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	out-of-order execution, with speculation
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by compiler

(Source: Computer Architecture, A Quantitative Approach, 4th edition: Figure 2-18)

Design a superscalar out-of-order MIPS pipeline that can issue four instructions every cycle. The MIPS instruction set is available from the Henessy and Patterson text book.

Your design should allow for the following:

1. Allow fetch, decode, issue and commit of two instructions every cycle as possible.
2. You will need to create a multi-ported register file.
3. Assume four ALUs and multiple copies of other processing elements as needed.
4. Allow out-of-order execution.

The pipeline will attempt to fetch, decode, issue and commit 2 instructions each cycle. The ideal achievable CPI is, therefore, 2.

Note that you will be required to implement Register renaming (cf. H&P, more materials available on request) and speculative execution with simple always-taken branch prediction.

Project III: Bitcoin Miner

Design, implement, validate and synthesize a ***Bitcoin Miner***.

First familiarize yourself with this gentle introduction to Bitcoin:

http://www.economist.com/blogs/babbage/2011/06/virtual-currency

Then read through a more technical introduction to Bitcoin from one of the core developers:

https://s3.amazonaws.com/gavinandresen-bitcoin/GavinAndresenCIATalk.pdf

Finally refer this paper for further technical questions.

http://bitcoin.org/bitcoin.pdf

Now you have all the requirements, your goal is to develop a master-slave system to accelerate bitcoin mining.

The "master" system will perform all networking related functions such as receiving a transaction block from the bitcoin network, performing peer-to-peer transaction etc.,

The "slave" processor, the bitcoin miner you will be designing, will perform the core hashing algorithm.

The core mining algorithm involves computing two rounds of SHA-256 hashes on a nonce and the transaction data block received from the host processor. Here are some specifications of the Miner for your reference:
Pseudocode: https://en.bitcoin.it/wiki/Getwork;
Java: http://code.google.com/p/bitcoinj/
C miner: https://github.com/jgarzik/cpuminer/blob/master/cpu-miner.c

There is even an FPGA/Verilog implementation of the bitcoin miner here:

https://github.com/progranism/Open-Source-FPGA-Bitcoin-Miner/

(Obviously you cannot use the source code there because your design target and performance requirements are completely different.)

Your design should be able to receive from, and transmit information to the host processor. Assume that the host and slave are connected with a 16-bit bus. The command and the data signals travel on the same bus; and data widths wider than 16 bits will be transmitted through multiple cycles.

Next, design and implement the core algorithm. A key design requirement is high performance; you will probably have to use pipelining and parallelism. You may want to use multiple SHA engines on a chip ("multicore bitcoin mining") to improve performance. To compute the number of cores that can be placed on chip, assume a die area of 100mm2 and 90nm technology libraries.

The SHA standard is available at the following URL:

http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

Finally validate the entire design.

Project IV: A Memory Controller

For background read the class notes on DRAM and DRAM memory controllers.

Implement the memory controller described in this paper:

http://dl.acm.org/citation.cfm?id=2337162