Catalog of Questions for the Final Exam

Pipeline/Optimization

- Since we didn't have any questions on the midterm and students were well-prepared, I plan to include this in the list of topics for the final.

Timing/Protocol

There will not be any questions on VHDL at the finals. As stated during the review, students should be able to devise Datapath + Controller from the specification of the components.

We are only interested in the design. Datapath + Controller. All the details of the components will be provided to you and we will expect you to provide the design of the circuit. We assume that every student in this class can implement this in FPGA or simulate using VHDL.
Focus only on the logic. If it's not clear to you, no worries. The final will provide sufficient details.

Always valid timing. Aside from sensors and games, give three other examples of where always valid timing is used

Combinational logic output,

Periodically valid timing. Aside from what was mentioned in the text, give three examples of where periodically valid timing is used.

Flow-controlled timing. Aside from what was mentioned in the text, give three examples of where timing with flow control is used.

Conversion to periodic timing. Design a VHDL design entity that acts as a receiver for an interface with an eight-bit wide data signal and ready–valid flow control and as a sender for an interface with periodic timing with a period of N = 5. Explain how you handle the case where your module is empty when the next period comes up.

Conversion from periodic timing. Design a VHDL design entity for receiving periodic (N = 10) eight-bit signals and outputting them to a ready–valid interface. Your module should include the ability to save up to two of the periodically valid signals if the output is not ready. You may drop the third such packet.

Fully utilized flow control. In the example of chapter 5 - slide 7, we designed a module that can be used to buffer a ready–valid flow-controlled communication channel. The problem with this, however, was that we were able to accept new data only every other cycle. Design a module that can accept a new value every cycle, enabling full throughput. You are not allowed to have any combinational paths from the downstream interface to the upstream interface (or vice versa). You will need two registers to store incoming data.

Credit-based flow control. Credit-based flow control is an alternative to ready–valid signaling. The sending module starts with n credits. On every cycle in which the module still has at least one credit remaining, it can emit an eight-bit data signal and signal valid. The receiver is guaranteed to capture this value. Sending valid data subtracts one credit from the count of remaining credits. A periodically valid (period of 1) signal from the receiver to the sender, creditRtn, is used to "return" credits back to the sender. Every cycle during which creditRtn is asserted, the sender increments the credit count. Design and write VHDL for this credit-based sending module. The inputs are an always valid eight-bit data signal, reset, and creditRtn. The outputs are the valid signal and data.

Serialization, I. Design a VHDL design entity to convert a 64-bit data signal with periodic timing (eight-cycle period) into a series of eight-bit signals with periodic timing (one-cycle period). You must store the input data, since it can change to unknown values when it is not considered to be valid.

Serialization, II. Repeat the previous exersice, but instead assume that the 64-bit input uses ready–valid flow control. The output interface also uses a ready–valid protocol, but at a frame granularity. When the output signals ready and the input is valid, the input sends all eight eight-bit packets in eight consecutive cycles. You are not allowed to have a purely combinational path from the upstream to the downstream interface (nor from the downstream to upstream interfaces). What is the maximum utilization of the output?

Frame and cycle-level flow control. Design a VHDL design entity that receives a serialized signal over eight cycles using frame-level flow control and acts as a sender for a serialized signal over eight cycles using cycle-level flow control.

Interconnect/Bus/Crossbar/Arbitration

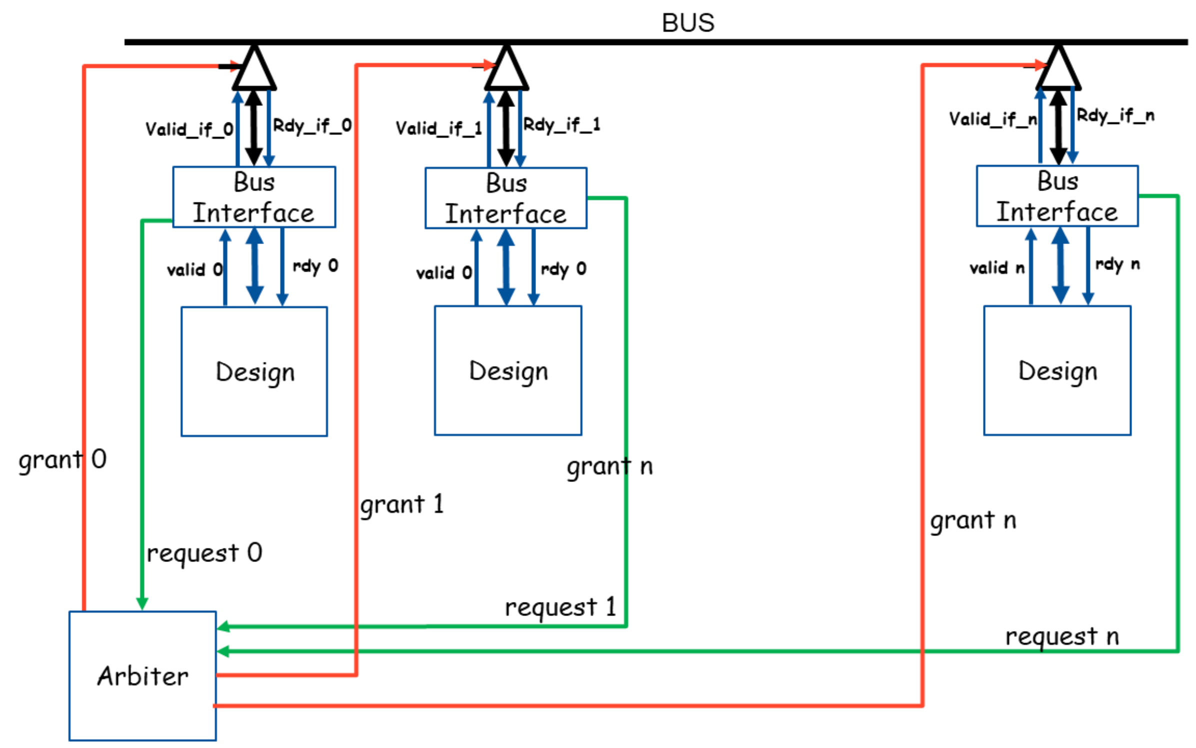
We are only interested in the design. Datapath + Controller. All the details of the components will be provided to you and we will expect you to provide the design of the circuit. We assume that every student in this class can implement this in FPGA or simulate using VHDL.
Focus only on the logic. If it's not clear to you, no worries. The final will provide sufficient details.

Bus Design:

Provide the design of a bus system using tri-state buffer.

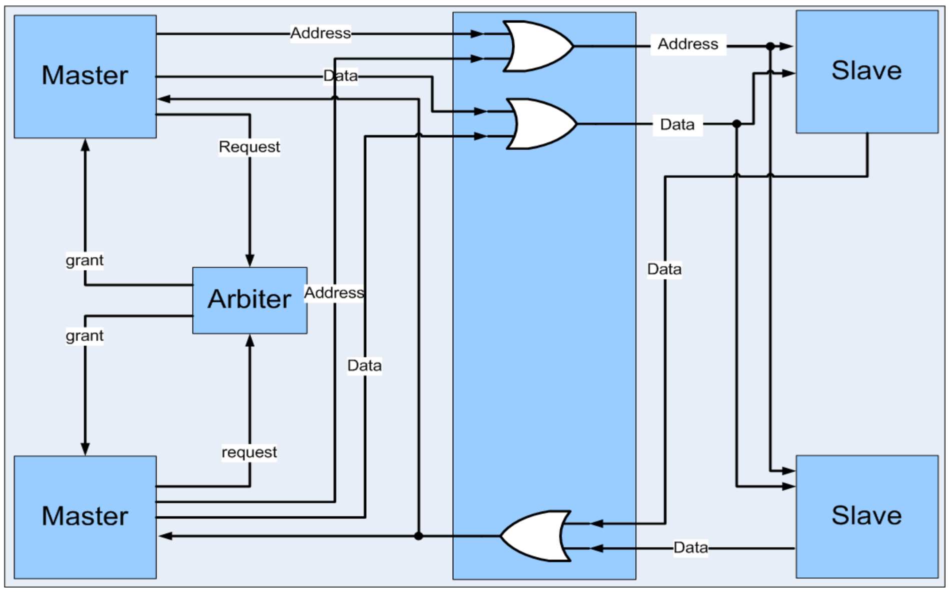
- Architecture
- Bus Interface
- Arbitration

You should understand and be able to explain every aspect of the figure below.
The meaning and function of every signal and component on the picture
The algorithm within the Arbiter that determine which component to select in case of simultaneous request is not of interest for the final.

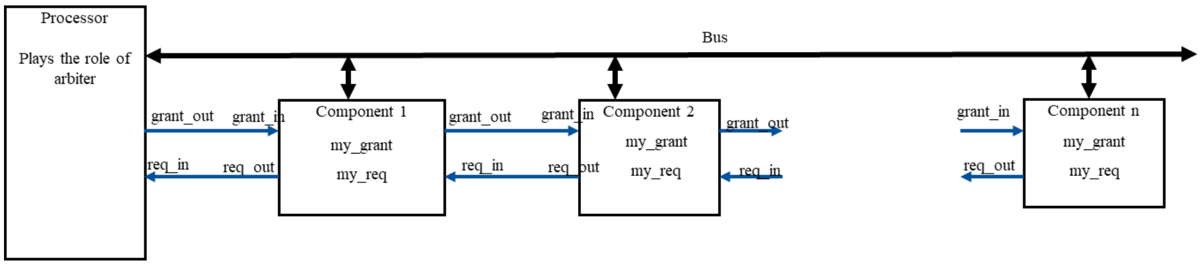


Repeat this operation for a combinational bus.

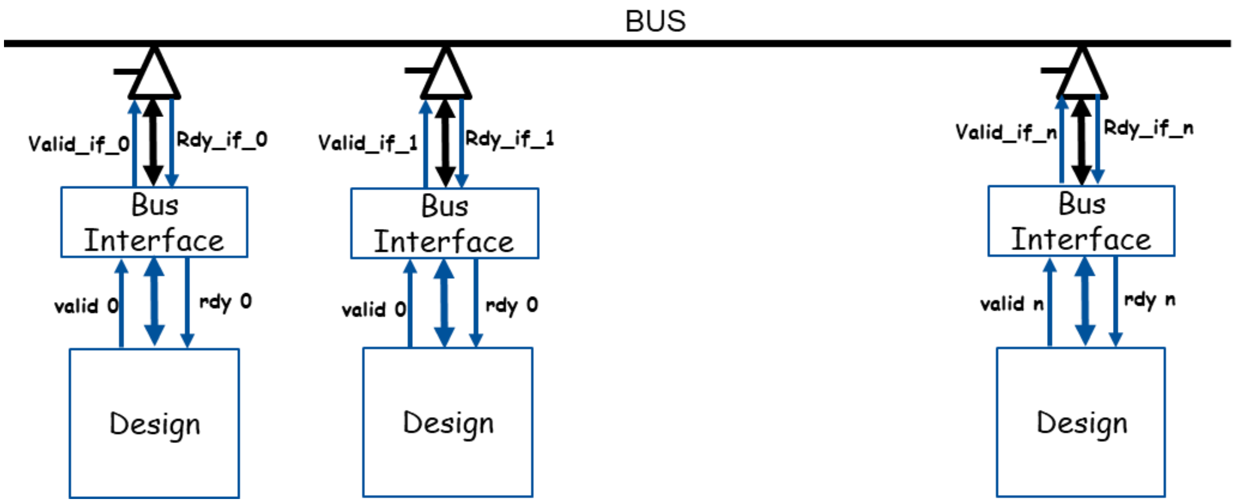
The picture below for the combinational bus should also be clear to everyone.
Explain where the valid and ready signal should be added for a bus system connecting multiple components following the ready-valid protocol



Daisy-chain bus arbitration. Design a controller and arbiter for a combinational daisy-chained bus. A daisy-chained bus does not have a centralized arbiter, but rather each controller makes a local request/grant decision. Controller 0 will always receive a grant and place its data onto the bus if it has a request. Controller 1 grants itself access to the bus only if controller 0 has not made a request, and so forth. Controller N will get the bus only if all N − 1 downstream controllers have no requests.



Distributed bus arbitration. Write a controller to carry out distributed bus arbitration. During each round of arbitration (over multiple clock cycles), each controller with a request puts its priority onto a bus that ORs together all signals. On the first round, if the MSB of the bus priority is greater than that of a given controller, that controller drops out of participation. This process is then repeated using the MSB −1 bit and all remaining requesters. At the end of arbitration, only one controller will remain and become the bus master.



4 × 4 crossbar. Write the VHDL to implement a 4 × 4 crossbar switch with full flow control. Use the same input and output signals as in Figure 24.5, but with two more controllers.

Multicast crossbar. Design a 4 × 4 crossbar that supports multicast messaging. Each input can request one or more outputs, but arbitration must be done as all or nothing. That is, an input is either granted to send to all outputs or can send to none at all.

Serialized crossbar. Modify a 2 × 2 crossbar to allow the serialized transport of a 20-bit payload. The crossbar wires should be only four bits wide and should perform arbitration and flow control on the head only once for each packet.

Buffered crosspoints. Design a crossbar to have n2 buffered crosspoints. On each cycle, every input will write into the buffer that connects that input to the desired output (provided the buffer is not full). The output channels then arbitrate between the input crosspoints with requests, popping one of them and outputting the data.

VHDL implementation of a simple router. Write the VHDL code for a simple router for use in a mesh network with one processing element per router. Your router should have a single client port with ready–valid flow control in both directions and ports for channels in four directions (west, east, north, and south) also with ready–valid flow control. Each of the five inputs to your router should provide double buffering so that if the next channel is not immediately available the packet can be buffered without requiring a combinational path to the previous router. Assume that the entire route is encoded in the address field of the packet, with three bits per hop specifying the port for that hop.

Memory

Memory Organization

- Given an input address, data size and memory primitive, be able to assemble the target memory.
- Example: A microprocessor interfaces a memory using 24-bit address and 48-bit data bus. To assemble the memory, a computer architect uses a library that only provides 4M x 8 memory elements. Provide the architecture design of the whole system.

See class slides (chapter 5-Integration) for the banking + slicing organization

- How many memory modules do we need in a row to store an entire word from the microprocessor?
  48 bit data but only 8 bit data memory available → 6 memory elements in parallel per row.
- How many rows do we need?
  1 memory element stores 4MB (2^{20}x4 = 2^{22}) rows.
  With 24bit the memory size needed is 2^{24}
  Number of row = total number of rows needed/number a row per memory element = $2^{24}/2^{22} = 2^2$= 4
- How is the address divided?
  Bank Selection | Row select | column selection| byte selection
  We may not need everything in this division, depending on the memory configuration
  2 | 20 | 3 | 0

  Show the whole architecture on a graphic
  The upper 2 digit are inputs to a 2-4 decoder to select the corresponding row.
  The next 20 bits are used to address a row within a memory element.
  Because we are addressing all the row simultaneously, we use the next 3 bits to select the column. if we are required to transport column sequentially. In the present case, the processor uses words

of 48 bit. There is no need to father divide the lower bit. Once we have the row address within a bank, we read everything in parallel.

Bandwidth: Number of transactions (read or write) pers second
If the decoding time for the memory takes 15ns and that it takes 60ns to transport the data from memory to CPU. Compute the bandwidth (number of memory transaction per second) of this organization.

Assume that the bus between the memory and the processor is 16-bit wide and the transport latency does not change. What is the bandwidth with this configuration?

Memory addressing. For each of the following memories, state how many bits are needed in order to address the full capacity. Also explain which bits are used for byte selection, bank selection, and word selection. Assume byte addressing, and that the bank selection is done with the bits just after the byte select bits.

1. One array with 2000 32-bit words.
2. Eight bit-sliced arrays, each with 1000 16-bit words.
3. Sixteen banked arrays, each with 512 128-bit words.
4. Eight banks of 16 bit-sliced arrays, each array has 1000 64-bit words.

VHDL SRAM Implementation. Using the RAM primitive of the figure below write the VHDL to implement the following:

1. a memory of eight bit-sliced arrays, each with 1024 16-bit words.
2. a memory of 16 banked arrays, each with 512 128-bit words. Only the needed bank should be activated.

```
-- RAM of parameterized size and width
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity RAM is
  generic( data_width: integer := 32;
           addr_width: integer := 4 );
  port( ra, wa: in std_logic_vector(addr_width-1 downto 0);
        write: in std_logic;
        din: in std_logic_vector(data_width-1 downto 0);
        dout: out std_logic_vector(data_width-1 downto 0) );
end RAM;

architecture impl of RAM is
  subtype word_t is std_logic_vector(data_width-1 downto 0);
  type mem_t is array(0 to (2**addr_width-1)) of word_t;
  signal data: mem_t;
begin
  dout <= data(to_integer(unsigned(ra)));

  process(all) begin
    if write = '1' then
      data(to_integer(unsigned(wa))) <= din;
    end if;
  end process;
end impl;
```

DRAM timings, I. Assume a DRAM has 5–5–5–12 timings. Addresses are eight bits, with the upper four bits being row-select and the lower four column-select. Answer (1) and (2) for the following address stream: 01, 02, 03, 10, 20, a3, b3, 04, b1, b2.

1. What is the total delay? (You must start and end with all rows precharged.)

Command	Time
Activate R0	5
Read C1	5
Read C2	5
Read C3	5
Precharge R0	5
Act R1	5
Read C0	5
RAS: Wait to PC	2
Precharge R1	5
Act R2	5
Read C0	5
RAS: Wait to PC	2
Precharge R2	5
Act Ra	5
Read C3	5
RAS: Wait to PC	2
Precharge Ra	5
Act Rb	5
Read C3	5
RAS: Wait to PC	2
Precharge Rb	5
Act R0	5
Read C4	5
RAS: Wait to PC	2
Precharge R0	5
Act Rb	5
Read C1	5
Read C2	5
Precharge Rb	5
Total	130 cycles

2. If you can rearrange the requests at will, what is the new delay?

Command	Time
Activate R0	5
Read C1	5
Read C2	5
Read C3	5
Read C4	5
Precharge R0	5
Act R1	5
Read C0	5
RAS: Wait to PC	2
Precharge R1	5
Act R2	5
Read C0	5
RAS: Wait to PC	2
Precharge R2	5
Act Ra	5
Read C3	5
RAS: Wait to PC	2
Precharge Ra	5
Act Rb	5
Read C3	5
Read C1	5
Read C2	5
Precharge Rb	5
Total	106 cycles

DRAM timings, II. Compare a DRAM with an 800 MHz I/O clock and 8–8–8 timings with one with a 1 GHz I/O clock and 12–8–8 timings (we are ignoring tRAS in this exercise). Which is faster for the following access patterns?

1. A series of completely random addresses that are always to different rows.
2. A series of addresses that are to an open row 99% of the time.

What percentage of accesses need to be to an open row in order to get equal performance?

Verification

- What is difference between verification and testing?
  - Verification is performed on a function to check its correctness
  - Testing is done on the implementation. Chip or FPGA to ensure that there were no fabrication defects. In the case of FPGA, to ensure that the configuration of the FPGA is fault free.

- Name three main challenges in verification?

  - Design complexity

  - Understanding the specification

  - Correct coding

  - Corner cases are usually very difficult to find

  - Time to market pressure. We want to bring the product to the market within a given deadline

- What is the difference between functional and formal verification?
  - Functional simulation: Simulation-based and static based
  - Formal verification: Use mathematic formula to prove or disprove assertions
- Why is verification so important? provide 3 reasons?
  - Preventing recalls
  - Designs are too complex to think of everything.
  - Saving money. Non verified designs extend the development cycle.
- What are the components of a functional verification? Show on a graphic how these components are used and explain.
  - Stimuli generator (sequencer), with randomization capability
  - Monitor to check inputs and outputs sent to the DUT
  - Scoreboard (checker) to verify that the DUT is working correctly. Contains a model of the DUT-design
  - Coverage to check that all cases have been produced and verified
- What is a testbench?
- Is a testbench needed in formal verification?
  - No. Just assertion and model of the target function/device/component
- What is the difference between event-driven simulation and cycle-based simulation. Explain their advantages and drawbacks.
  - Event-driven: All events (changes on input signals) are propagated through the design. New event's resulting from change at the output are scheduled and propagated until the outputs are stable. Then the time is advanced. This process continues until the simulation time is reached. Accurate, but very slow
  - Cycle-based simulation: All events are evaluated within a clock cycle. New events resulting from changes at the output are not evaluated until the next clock cycle. Only the last event in a cycle count. Very fast but less accurate
- Given a processor with 32 32-bit register capable of performing the following operations;
  - add \$a, \$b, \$c
  - sub \$a, \$b, \$c
  - beq \$a, \$b Label
  - load \$a, (i)\$b

Provide a plan to perform verification of this unit. Your plan must include:

  - what needs to be verified?
    The minimum is to verify that all operation perform correctly. So add, sub, beq load.
    We may also perform the coverage on the internal registers of the processor. To make sure that all registers provide correct value to the ALU and that results are correctly stored
  - how the verification will be conducted. Testbench components?
    Use a sequencer (stimuli generator) that will drive the instructions randomly, and for the instruction, also drive the input data randomly, and define the destination registers randomly. Randomization must bee constrained to only 5 instructions (encoded on 3 digits). With no randomization, all 08 cases will be considered

  - how do we know that the verification was successful?
    We monitor the input to the design and send the input to the coverage component. We use bins to represent the case we want to count. An instruction for instance will have it's own bin. We then count the number of cases in a bin. The verification is complete when all our cases are covered. The values in the bins reflect our expectation.
  - how to make automatize the varication process
    Develop the sequencer, the scoreboard, the monitor and the coverage and let them run automatically until the stop conditions are reached.

- What does BIST stands for? Explain the need for BIST in the testing process.

Misc.

- What does BIST stand for? Explain the need for BIST in the testing process.