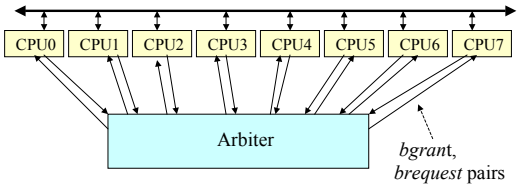
Sim #3 – 8 CPU + Arbiter Simulation

- Simulations can be used to obtain quantitative results for values that have no closed form solution or which are difficult to predict
- Will use an 8 CPU + Arbiter over a shared bus simulation as the target of Sim #3.



Definitions

- **IO transaction** – CPU asserts *breq*, is granted the bus by the arbiter, and assumes mastery of the bus by asserting *bbusy*.
- **Transfer size** – the number of clocks *bbusy* is asserted during an IO transaction.
- **Total Clocks** – the total number of clocks in the simulation
- **Bus Latency** – the number of clocks from assertion of *breq* to assertion of *bgrant* by the arbiter. For this bus protocol, at least 2 clocks with no bus contention.
- **Bus Utilization** – the ratio in clock cycles for (bbusy='0')/(Total Clks) * 100%. Note that this number can never equal 100% because it always takes at least one clock to change bus masters.

Fixed Priority vs Round Robin Priority

- A priority scheme refers to the method for selecting a CPU in the case of simultaneous bus requests
- A fixed priority scheme always uses the same priority based on bus request#
  - The arbiter in this simulation assigns CPU#0 the highest priority, CPU#7 the lowest
  - Disadvantage of fixed priority is that the lowest priority CPU can starve in the presence of high bus contention
  - Advantage is simplicity
- A round robin scheme rotates priority after every IO transaction
  - Idea is that each CPU has equal time at having the highest priority

Overlapped vs Non-overlapped Bus Grant

- For a non-overlapped bus grant approach, the arbiter only asserts a bus grant line once a previous CPU has released the bus
  - CPU does not have to monitor bbusy line, if granted the bus then the bus is free
  - Takes longer to hand off the bus between bus masters
- For an overlapped bus grant approach, the arbiter will assert negate the bus grant of the current master, and assert the bus grant of the next master, while the current master has the bus
  - CPU has to monitor bbusy line to see when bus is free
  - Minimizes hand off time between bus masters
- This simulation will always use the overlapped approach.

A Question

- For transfer size = 8 clocks, at what bus utilization does the difference in bus transfers between the highest and lowest priority CPUs exceed 20% in a fixed priority scheme?
- To answer the above question, need to simulate the system at different levels of bus utilization
  - The more IO requests a CPU makes, the higher the bus utilization
- Must measure the bus utilization for a fixed number of clocks
- Must record the number of IO transfers that each CPU makes during the simulation
- The Sim#3 assignment lists other questions that must be answered

CPU, Arbiter Model Generics

- The ZIP archive attached to the lab contains the arbiter, CPU, testbench, configuration models
- Arbiter generic ROUND_ROBIN controls whether rrobin or fixed priority scheme is used
- CPU generics:
  - RND_SEED – a number between 1 and 50 that is used to select a starting random seed value contained in the *rnd2* package.
  - CPU_ID – identifies this CPU and is the number placed on the address bus when this CPU is bus master
  - CLK_MAX – when the total number of clocks seen thus far equals this value, the CPU should halt all activity and assert its *active* output to the 'Z' value. The active signal in the testbench has a weak pullup ('H') on it – when this signal transitions from '0' to 'H' all CPUs have stopped.

CPU *request_rate* Generic

- The *request_rate* generic will be used to control the number of IO requests a CPU makes – the higher this number, the more IO requests the CPU should make.
  - The more IO requests, the higher the bus utilization
- The CPU model has a finite state machine – the *local* state represents the clocks in which the CPU is not making an IO request
  - The more clocks spent in the *local* state, the fewer IO requests that are made
- Declare a boolean array called *req_array* that has 2000 elements
  - For each clock spent in the local state, increment a pointer (*index*) into req_array
  - If req_array[index] = TRUE, then make an IO request
  - Initialize *req_array* such that *request_rate* number of values are TRUE, and use a random number generator to pick these locations in *req_array*.

rnd2 package

The *rnd2* package contained in the ZIP archive contains a set of functions/procedures for generating random numbers.

The code below shows how to use this package to generate 10 random numbers between 0 and 999 inclusive

```
process
   variable bound_h: Real := 999.0;
   variable bound_l: Real := 0.0;
   Variable rnd_rec: rnd_rec_t;
   variable ll: line;
 begin
   rnd_rec.distribution := rnd_uniform_d;
   rnd_rec.seed := rnd_seeds(5); --random seed
   rnd_rec.bound_l := bound_l;
   rnd_rec.bound_h := bound_h;
     for i in 0 to 9 loop
       Rnd_Random(rnd_rec);
       write (ll, integer(rnd_rec.rnd));
       writeline(OUTPUT,ll);
     end loop;
   wait;
```

Record type passed to *rnd_random* proc. Fill in the bounds before first call.

Init random seed using a value from *rnd_seeds* array; 50 seeds available

Generate random number, uniform distribution.

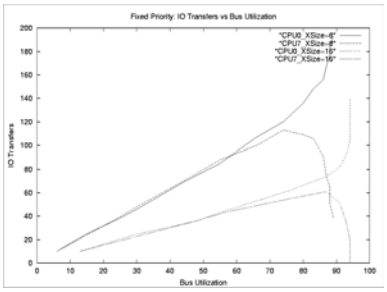
Collecting Statistics, Printing Results

- You will need to add a VHDL package of your own that defines the shared variables needed to collect any statistics required to answer the questions
- Also need to print out a report once the specified total number of clocks have been reached (I will use these numbers as a rough sanity check on your model)

```
# CPU0 TClks: 10000, TIOs: 25, TLatency: 57, LatPerIO: 2.280000e+00
# CPU1 TClks: 10000, TIOs: 24, TLatency: 78, LatPerIO: 3.250000e+00
# CPU2 TClks: 10000, TIOs: 24, TLatency: 55, LatPerIO: 2.291667e+00
# CPU3 TClks: 10000, TIOs: 24, TLatency: 64, LatPerIO: 2.666667e+00
# CPU4 TClks: 10000, TIOs: 24, TLatency: 59, LatPerIO: 2.458333e+00
# CPU5 TClks: 10000, TIOs: 22, TLatency: 107, LatPerIO: 4.863636e+00
# CPU6 TClks: 10000, TIOs: 24, TLatency: 63, LatPerIO: 2.625000e+00
# CPU7 TClks: 10000, TIOs: 24, TLatency: 57, LatPerIO: 2.375000e+00
# TransferSize: 8 ReqRate: 5 %BusUtil: 15%
AvgIOs: 23 AvgTotalLatency: 67 AvgLatencyPerIO: 2.913043e+00
```


Plots

By plotting Average IO latency vs Bus Utilization, and IO transfers versus Bus Utilization can answer the questions.



May need to change scale on Bus Utilization axis to get higher resolution in some cases.

Sanity Checks

- Please do simple sanity checking on your statistics
- Bus Utilization < 100 %
- For low request rates, there is little bus contention, so:
  - Latency per IO should be close to 2
  - Number of IO transfers made by each CPU should be close to (Total Clocks)/2000 * req_rate
  - Bus utilization will be close to (Number of IO * Transfer size * #of CPUs)/Total_clocks * 100%

Regression Testing

- Multiple simulations runs have to be performed with different values of *request_rate*, *transfer_size* and priority scheme.
- This is known as regression testing, and it should be automated to save time
  - Automation usually done via an external scripting language such as Perl
- The zip archive contains a Perl script called *sim3_sol.pl* that can be used for this.
  - Look at the comments in the perl script for usage directions
  - The script reads a template file called *sim3/cfg_tb.template* that contains place holders for model generic values and produces a new *cfg_tb.vhd* file with actual values substituted for model generic values
  - Number of simulation runs is determined by parameter specification in *sim3_sol.pl* - feel free to modify this script to suit your needs or write your own in your favorite scripting language.

Report, Model Checkoff

- Include your graphs, answers to questions in a file called 'report.pdf'.
- If you need to expand portions of the graph to get the required answers, then do so.
- I don't expect answers past one decimal point (ie. 3.5). I do expect answers with at least this fidelity ("about 4" is not acceptable).
  - You need to illustrate either via the graphs or model numerical output how you got your answers.
  - If you give me an answer without justification, I will count it as wrong.
- I will run your simulation with my own values for request_rate, transfer_size, priority scheme and examine your model output.
  - I don't expect your numbers to match mine exactly, but they should be reasonably close.