CS 254 (Digital Logic Design Lab) Project Report

Project Topic : Memory processor interaction

Team : Scopemaxx

Team members:
Kapil Vaidya (140050006)
Mohit Vyas (140050015)
Govind Lahoti (140050020)
Kushal Babel (140050022)

High Level Architecture

The system consists of a memory and processor where the processor is four times faster than the memory. The processor gets the requests of words and address from the lane interface and then processes it. The processor caters to the requests and then sends them over to the memory through the memory interface. For effective communication, there are different control signals between the processor and the memory. To make up for the mismatch, we have to use more number of I/O lines and FIFOs instead of a simple communication over a wire. The memory then writes the received word in the bank specified by the address after address is verified by the memory.

Things to take care of **(Tricky Parts)**

Load Balancing
Since the buffer space of different FIFOs corresponding to four users is limited, system should process the requests from different users in such a way that the size of FIFOs, in which requests are stored, doesn’t overshoot.

Since different users operate with different lane speeds and lane widths, there is high possibility of occurrence of above situation. System handles this by giving higher preference to the user request FIFO having higher requests yet to process. To implement this, a modified Arbiter is designed (instead of a standard round robin arbiter) which grants requests to the FIFO having higher size.

The efficiency of system is affected by the number of I/O lines and the number of bits the processor processes in each clock cycle. There is a tradeoff between minimization of I/O lines and the number of requests processed in a given time. The parameters of the efficiency of the system are idle time of Lanes, idle time of I/O lines and amount of words processed.

Processor-Memory Synchronization
Since the clock speed of memory is four times slower than that of processor, the synchronization of data transfer between memory and processor is a crucial part. This is achieved by adding a memory-processor interface which buffers the high-speed data from processor and transfer it to memory appropriately and vice-versa.

Work conserving
NO I/O lines(between processor and memory) should be kept idle, if there is a request pending at any of the user lane.

Block level structure

Our logic will have two main blocks:

1. Processor
2. Memory

Processor will in turn contain following sub-blocks:

1. Lane Interface
2. Memory Interface
3. Arbiter
4. Main processor

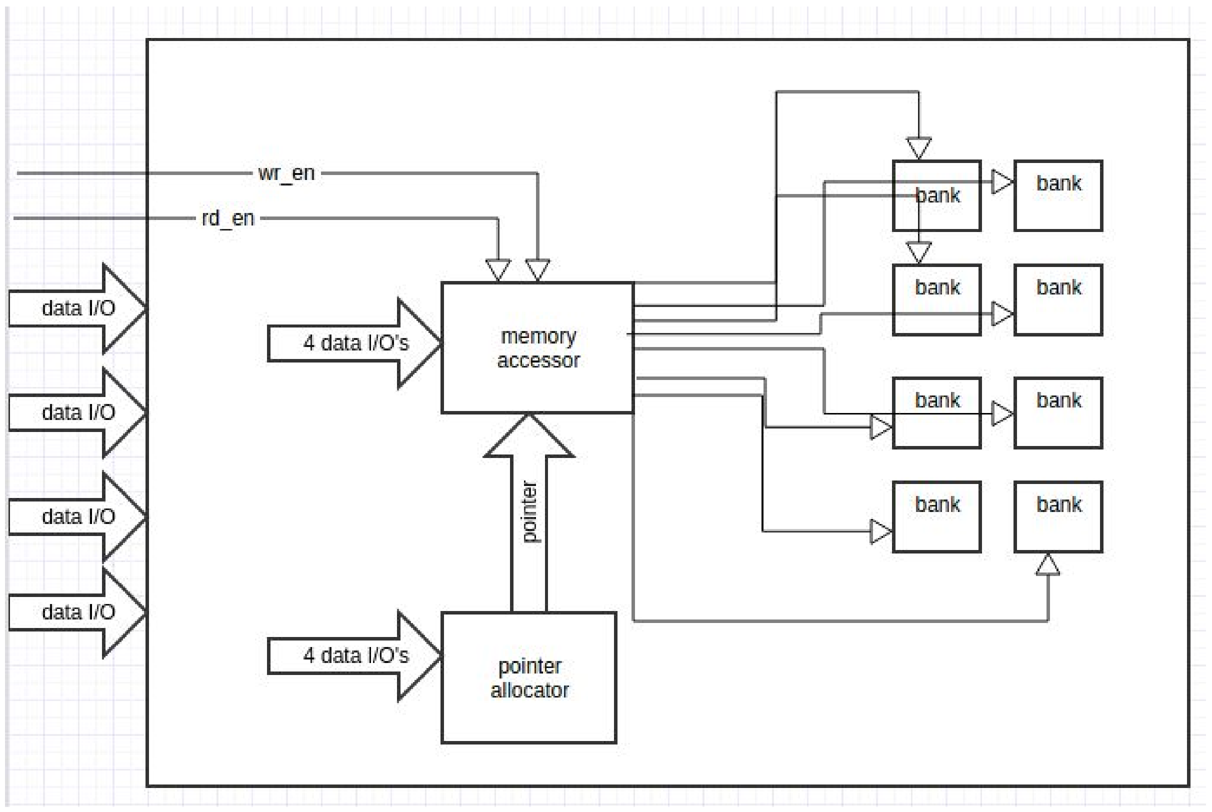
Memory will in turn contain following sub-blocks:

1. Memory accessor
2. Random number generator

Datapaths (previous & revised)

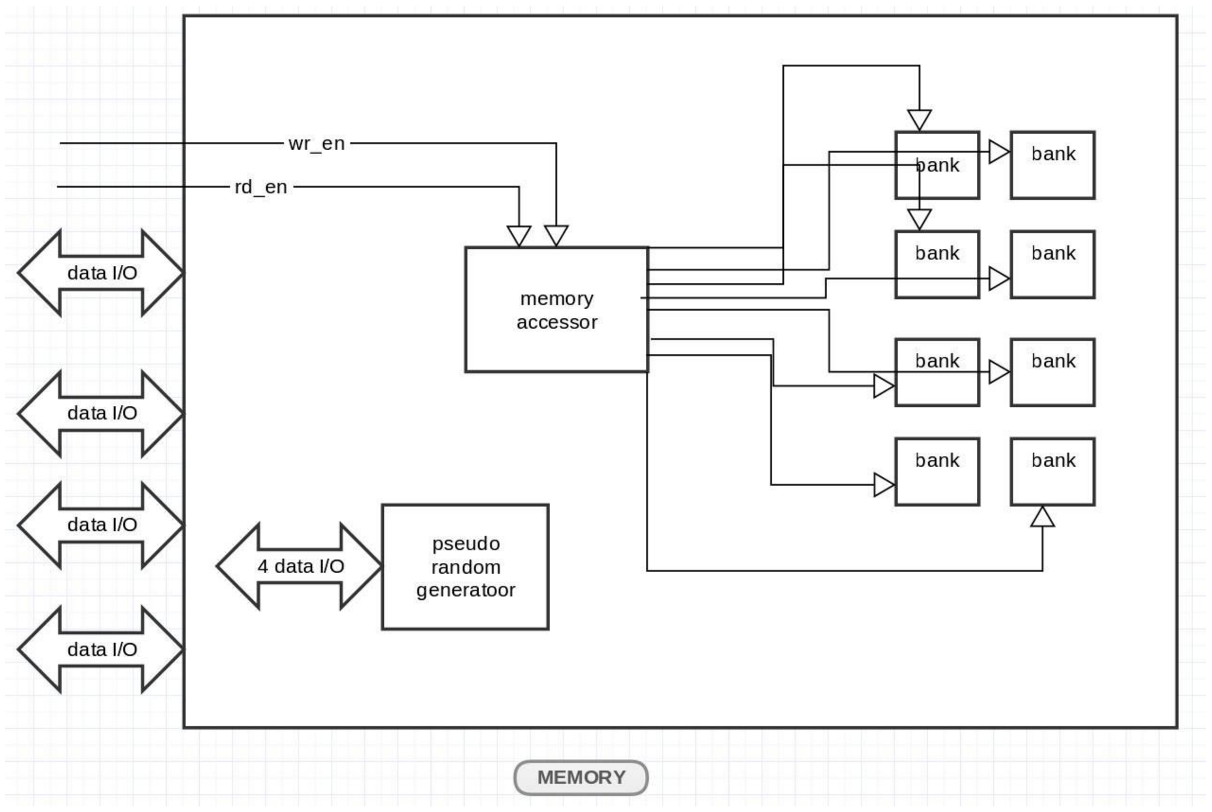
Memory

PREVIOUS MEMORY DATAPATH



REVISED MEMORY DATAPATH

(AFTER INSTRUCTOR FEEDBACK)



Sub-blocks of memory

1. Memory accessor

Ports -

The memory accessor has several inputs, namely pointer, write enable and read enable signals, and also the I/O lines which act as inout.

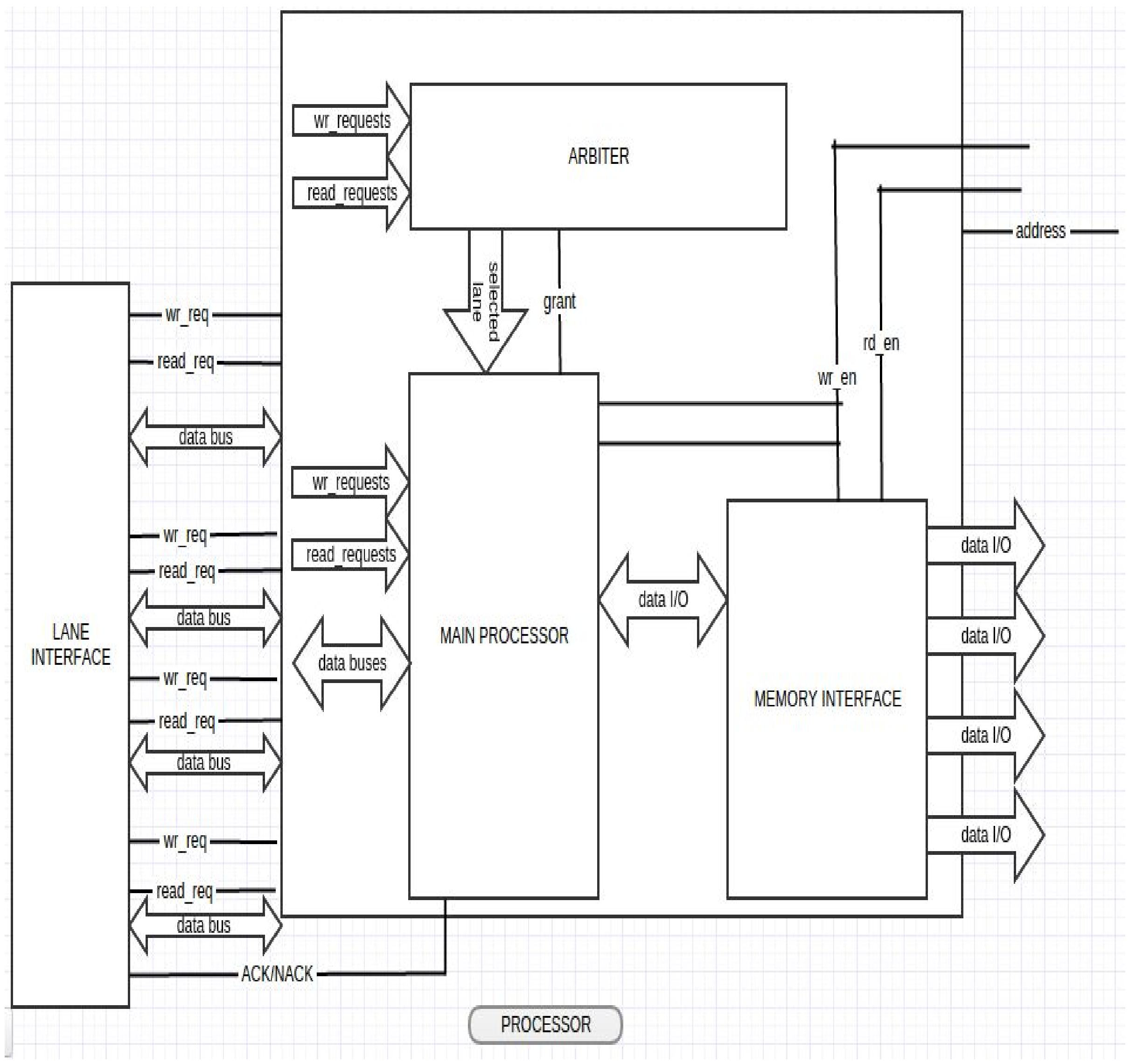
Working -
Suppose the write enable line is high, block checks if the location has already a word in it and sends appropriate response positive or negative.Then,memory keeps then the block keeps writing the data received through the input line in the address specified by the pointer input.
If the read enable signal is high, then block checks if the location has already a word in it and sends appropriate response positive or negative.Then,memory keeps then the block keeps writing the data received through the input line in the address specified by the pointer input.
It does not do anything when neither of the control line is high. The block has output lines to all the memory banks.
The block will have to maintain some internal variables for keeping track of the position(in the word) of the bits being written or read.

2. Random number generator

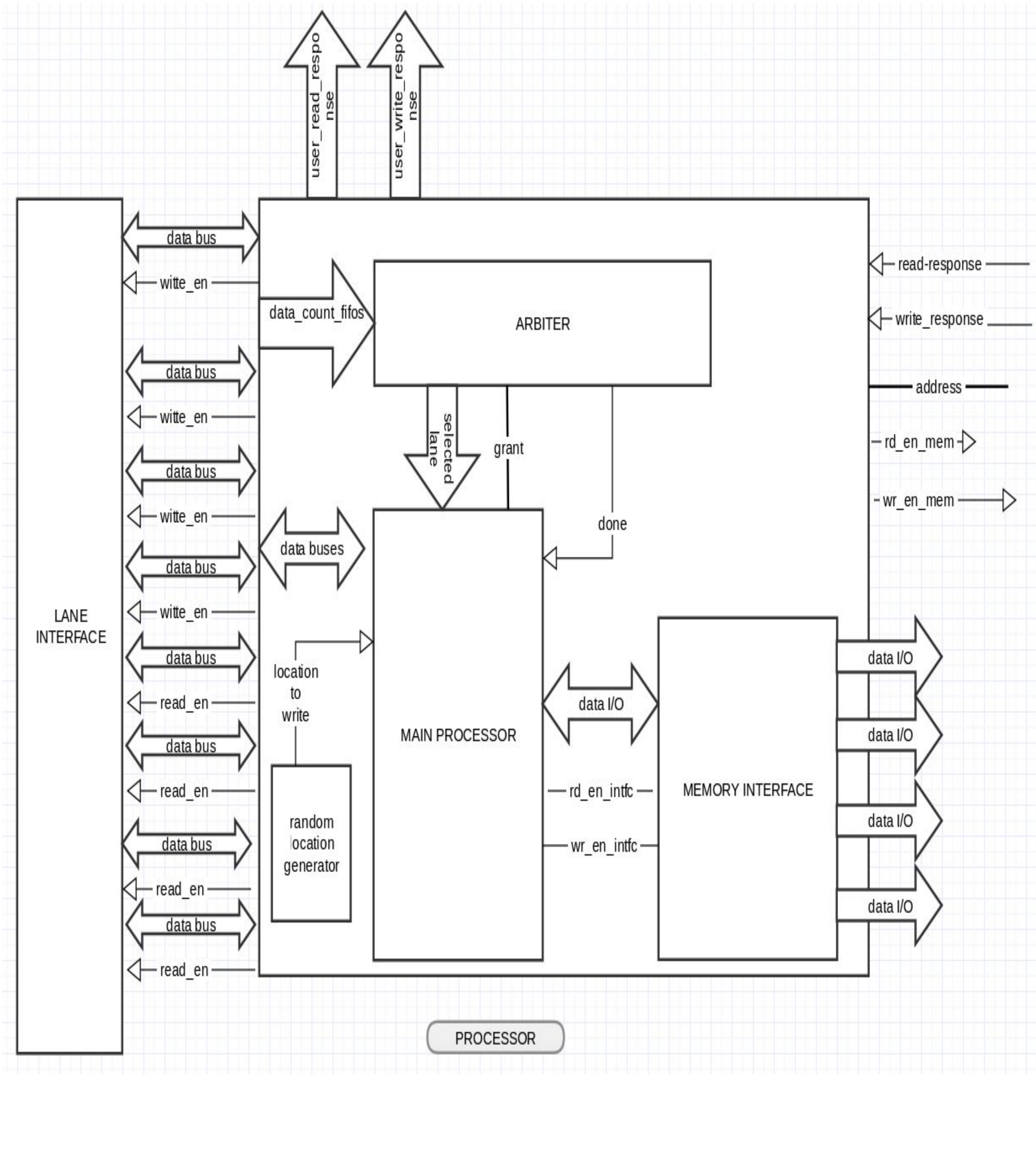
This block is used to generate pseudo random words. It will be used by memory as explained in code structure section.

Processor

PREVIOUS PROCESSOR DATAPATH



UPDATED PROCESSOR DATAPATH
(AFTER INSTRUCTOR FEEDBACK)



Sub-blocks of processor

1. Arbiter

Ports:-

The arbiter receives the fifo size signals from the Lane interface, also gnt signal from the processor.The arbiter sends out select lanes and done signal.

Working:-

Arbiter decides which of the four lanes' request should be granted according to the size of fifo. The arbiter starts working when grant is high. The output is four select lines to the processor which convey the lane number whose word should be sent or fetched and also a done signal which indicates when arbiter has decided which request to process.

Eg, Select_lane="01001" means process write request (select_lane(4)) of 2nd lane(select_lane(1));

2. Main processor

Ports:-

The main processor receives the lane number from the arbiter whose request should be processed. It also receives the read enable and write enable signals, and the word in the suitable format corresponding to all the four lanes through the lane interface.

Working:-

When the request is write type,a random pointer is generated and is provided to the memory. The length and the actual word bits are then sent to the memory interface if response of memory corresponding to location is positive otherwise word is not sent.While, sending the word processor manages appropriate enabling of signals given to interface and memory . Appropriate response is sent to user regarding status of the word after word is written ["10" location is already filled,"11" word written successfully].

When the read signal is high,pointer is provided to the memory.After receiving response from memory regarding location[“10” location has not filled reading garbage,”11” word written successfully]. It receives the word chunks from the memory interface and keeps on building the word and sending to the interface.Processor properly manages the enabling of control signals to interface and memory. It then returns the word to the along with appropriate response to user.

As soon as, the entire word is sent or received the corresponding control line is made low. When processor has no request to process it makes its grant signal high. The read response and write response signals are essential to make up for mismatch of processor and memory.

3. Memory interface

Ports:-

32 bit inout wide bus from processor to send and receive 32 bits packets of a word from processor during a read or write request. wr_en and rd_en signals from processor to indicate the start of input and expecting output respectively from processor. Four 32 bit buses to memory to transfer data onto memory.

Working:-

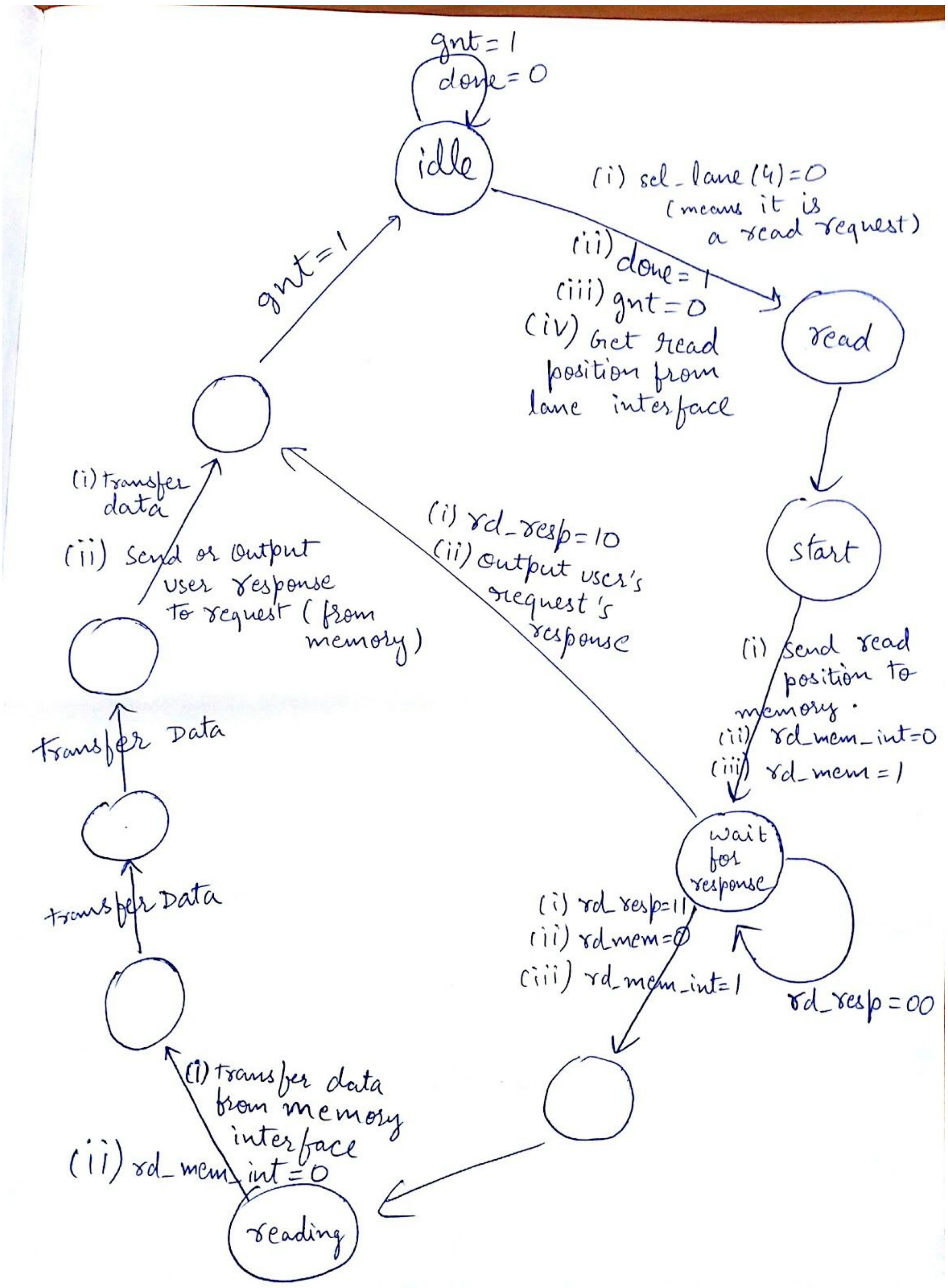
Its main working is to buffer data during the data transfers between memory and processor in order to compensate for the mismatch of speeds of memory and processor. While writing into processor, it takes four packets of size 32 bits each and send the word to memory whenever memory is ready.

Similarly, while reading data, it stores the data outputted by memory and pass it to processor at appropriate instant.

It plays an important role of synchronization between memory and processor along the half duplex link they are connected from.

State transition diagrams

Processor (for read requests)



Explanation of states

The dummy state has been introduced for grant signal to become low.
gnt-> processor sets it to 1 if it wants arbiter to choose a request

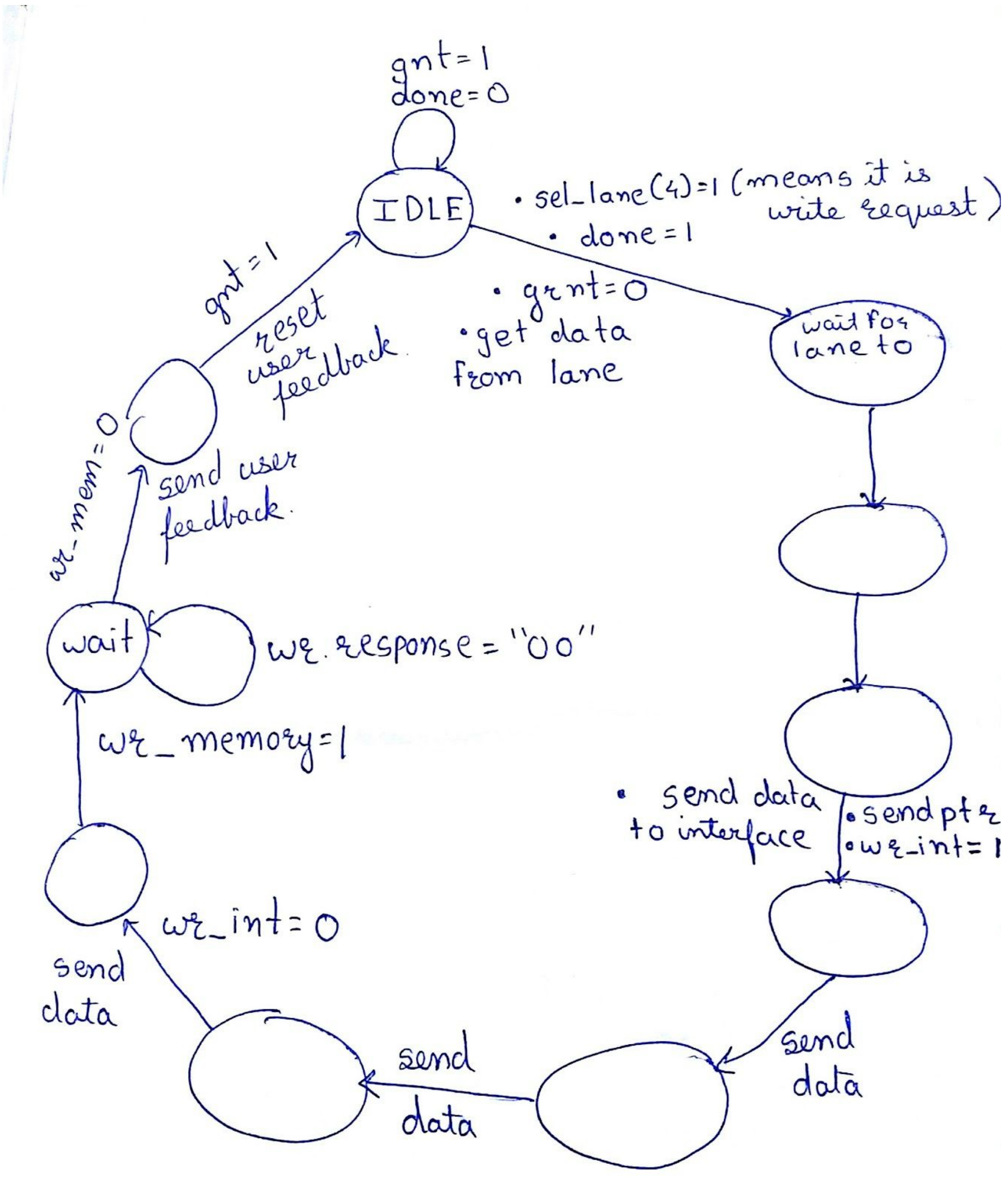
done-> done=1 signifies that arbiter is done.

sel_lane-> is a std_logic_vector that shows which request is granted by arbiter and whether it is read or write request. If sel_lane(4)=0 then its read request.

rd_mem and rd_mem_int -> std_logic which set to true if processor wants to read from memory or memory interface respectively.

rd_resp -> is 10 if it got an error while reading (i.e nothing was present at the position given by read request), and 11 if read was successful.

Processor (for write request)

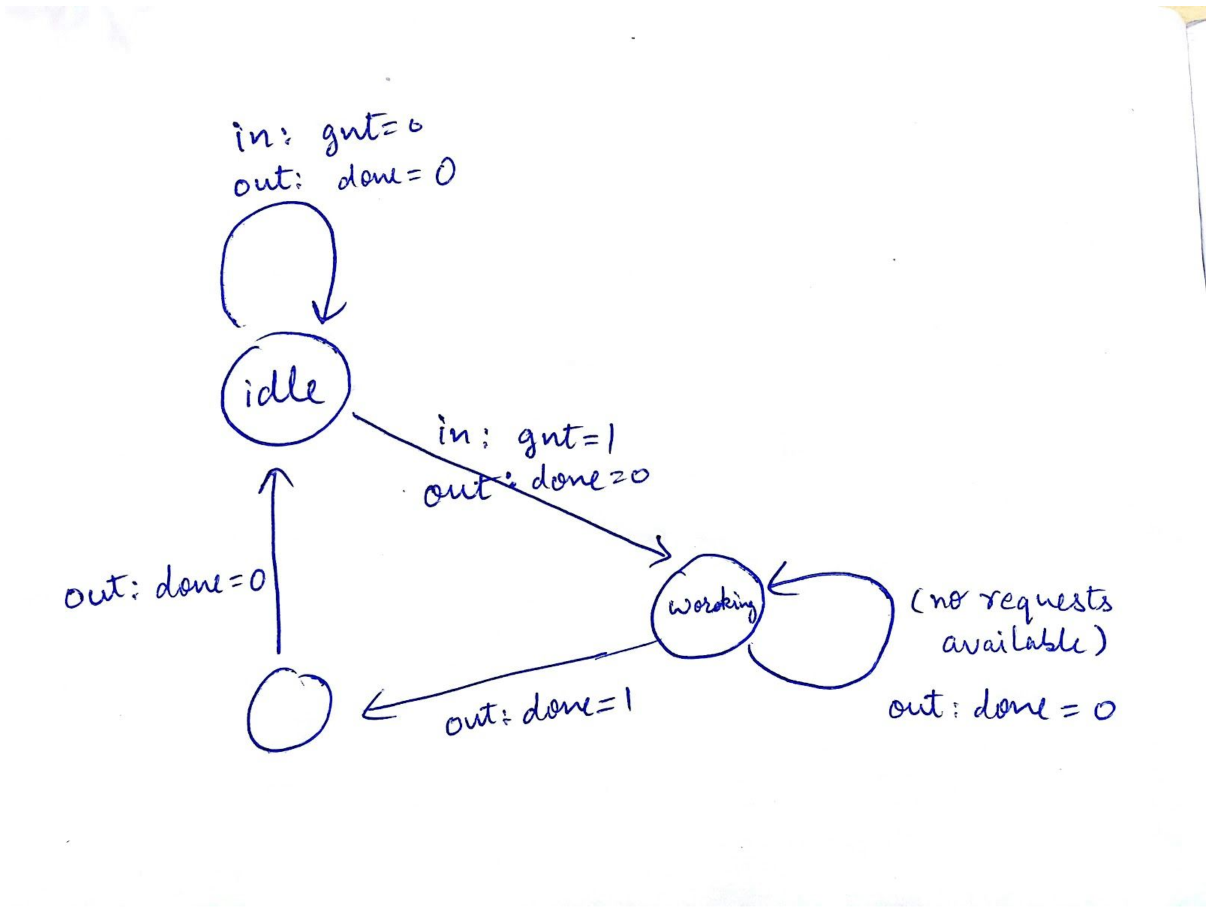


sel_lane-> is a std_logic_vector that shows which request is granted by arbiter and whether it is read or write request. If sel_lane(4)=1 then its write request.

wr_mem and wr_mem_int -> std_logic which set to true if processor wants to write from memory or memory interface respectively.

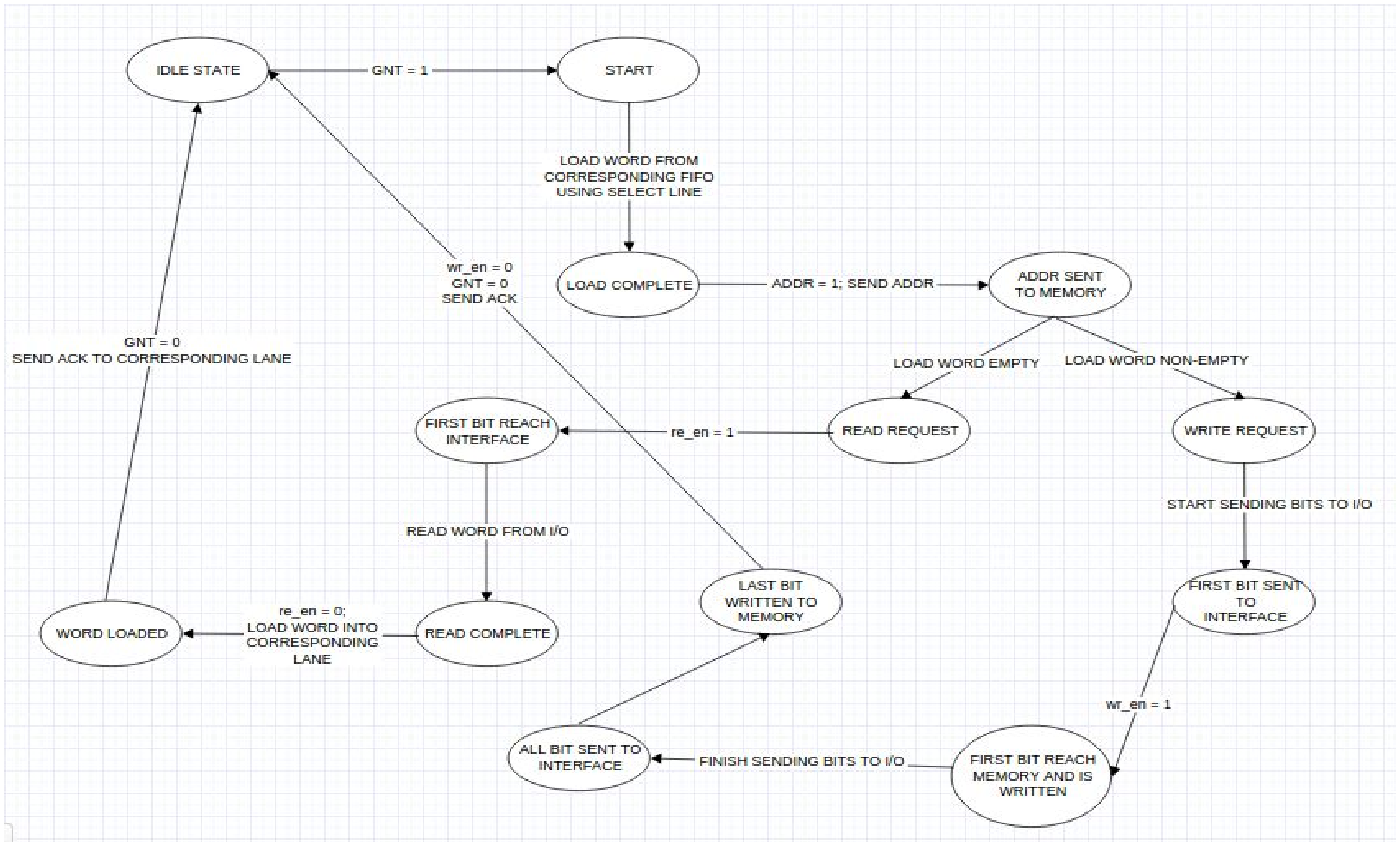
wr_resp -> is 10 if it got an error while reading (i.e nothing was present at the position given by read request), and 11 if read was successful.

Arbiter



An extra state has been introduced so that processor makes gnt=0 at the same time when done becomes 1 so that it can reach idle state (otherwise it will be forced to go in working state even if gnt=0)

First Stage State Transition Diagram:-


IDLE STATE
GNT = 1
START
LOAD WORD FROM CORRESPONDING FIFO USING SELECT LINE
LOAD COMPLETE
ADDR = 1; SEND ADDR
ADDR SENT TO MEMORY
wr_en = 0 GNT = 0 SEND ACK
GNT = 0 SEND ACK TO CORRESPONDING LANE
LOAD WORD EMPTY
LOAD WORD NON-EMPTY
FIRST BIT REACH INTERFACE
re_en = 1
READ REQUEST
WRITE REQUEST
START SENDING BITS TO I/O
READ WORD FROM I/O
LAST BIT WRITTEN TO MEMORY
FIRST BIT SENT TO INTERFACE
WORD LOADED
re_en = 0; LOAD WORD INTO CORRESPONDING LANE
READ COMPLETE
wr_en = 1
ALL BIT SENT TO INTERFACE
FINISH SENDING BITS TO I/O
FIRST BIT REACH MEMORY AND IS WRITTEN

Assumptions and constraints

1. Block-level assumptions

- Lane Interface provides the packet to the main processor in the desired format only. When writing, first it gives the length of word(7 bits) followed by the actual word (121 bits). If actual word is of length less than 121 bits, then remaining bits are filled with garbage values. When reading, packet contains just the address(9 bits).

2. System-level assumptions

- Word are maximum 121 bits long.
- Memory stores in the word in desired format only. For any location(128 bits) in the memory, first 7 bits are reserved for denoting the length, while the next

121 bits contain the actual data. (If word length is less than 121 bits, then garbage value is stored in the remaining bits)

3. Constraints

- Word length can’t be more than 121 bits long
- We have only 8*40 memory locations

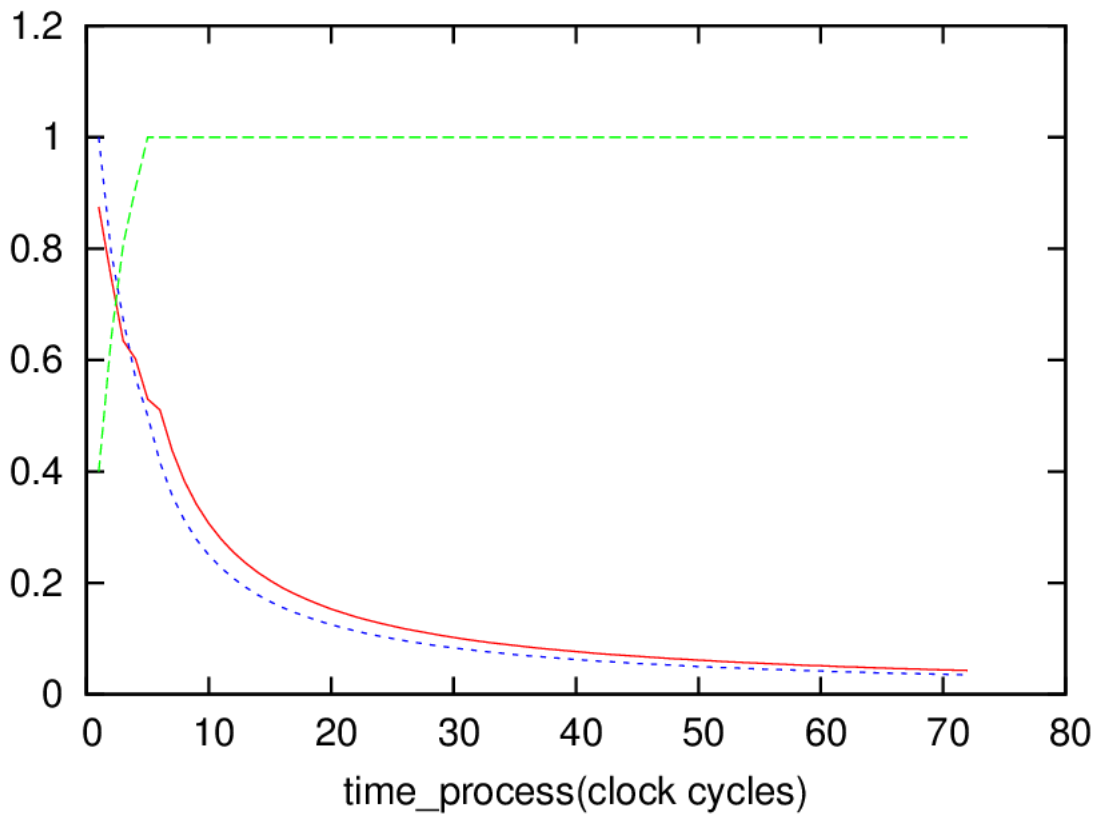
System Efficiency analysis

We need to analyse our system efficiency on various parameters.

1. Efficiency of Lanes :- Percentage of time lanes are busy in given time.
2. Efficiency of I/O lines :- Percentage of time I/O lines are busy in given time.
3. Number of request processed :- The total amount of requests processed by our system in given amount of time.

The analysis was done by writing a code which gives the time for which different lines are free by varying the number of bits processed by the processor in a clock cycle. The wordlength is considered as 72 for the analysis as a representative for words having length from 20 to 121.

Here is a plot against process time for a word.



Efficiency of lanes :- red

Efficiency of I/O lines :- green

Amount of words processed :- blue.

TR

We can see if time_process is less , so no of i/o lines used is large this leads to less efficiency of I/O lines.We can also see if time required for process is high i.e I/o lines are less this leads to less words processed and efficiency of lanes also decreases but efficiency of I/O increases as whole time have to send data.

Code Structure

Our code has following entities:-

1. **scopemaxx_top**
   (inside file scopemaxx_top.vhd)
   This entity is out top entities. It connects each individual entities

2. **lane_interface**
   (inside file lane_interface.vhd)
   This entity acts as lane interface. It contains 8 buffers (FIFOs) corresponding to read and write requests of the 4 lanes.

3. **smart_arbiter**
   (inside file smart_arbiet.vhd)
   This entity looks at the sizes of each buffer in lane interface and grants requests to the buffer with maximum requests pending.

4. **random_number_generator**
   (inside file random_number_generator)
   This entity generates a pseudo random number every clock cycle. This entity is used by test bench and memory (as discussed with Prof. Ashwin)

5. **processor**
   (inside file processor.vhd)
   It is entity that manages the lane_interface, memory_interface, arbiter and makes the proper communication between them. The state diagram of this component is explained in great detail below.

6. **memory_interface**
   (inside file memory_interface.vhd)
   It is a key entity that takes into account the different clock speed of processor(1 Ghz) and memory(250 Mhz). It manages the to and fro message transfer accordingly. 4 I/O lines of 32 bit width each have been used. Processor and memory communicate via this interface only.

7. **memory**
   (inside memory.vhd)
   As discussed with Prof. Ashwin, memory maintains a table, which t
   stores whether a particular memory location has been occupied or not. If the user attempts to write to already occupied memory

location, the operation would fail and appropriate signal is flagged to the user. Similar thing happens when user tries to read an invalid(out of range or unoccupied) memory location. Signals are flagged to the user for both operations for success and failure.
On reading a valid memory location, random word is returned by the memory.

Work Distribution

All the design level decisions was thought of and discussed by everyone. State machines too were prepared by common effort. Distribution of work was done only while writing the code. This is rough distribution.

1. Govind Lahoti : Implementation of memory, random no. generation.
2. Mohit Vyas : Implementation of lane interface and main processor.
3. Kushal Babel : Implementation of processor memory interface and lane interface.
4. Kapil Vaidya : Implementation of main processor and arbiter.

Test benches and documentation was too made by common effort.

-xxx-